Level Set and PDE Methods for Computer Graphics

Notes for SIGGRAPH 2002 Course #10
San Antonio, TX
July 21, 2002

Organizers

David Breen	California Institute of Technology
Guillermo Sapiro	University of Minnesota

Speakers

David Breen	California Institute of Technology
Ron Fedkiw	Stanford University
Stanley Osher	University of California, Los Angeles
Guillermo Sapiro	University of Minnesota
Ross Whitaker	University of Utah

Course Abstract

Level set methods, an important class of partial differential equation (PDE) methods, define dynamic surfaces implicitly as the level set (iso-surface) of a sampled, evolving nD function. In this course we present the underlying concepts, equations and numerical methods for these techniques and for related PDE methods. Practical considerations for implementing and employing level set/PDE methods for computer graphics will be discussed. These include developing a level set library and techniques for importing conventional geometric models into it. We will show how to apply level set/PDE methods to a variety of graphics applications, including image inpainting, pattern formation, 3D curve computation, 3D shape reconstruction, image and shape morphing, dynamic visibility, surface editing, and fire and water simulation.

Prerequisites

Knowledge of calculus, linear algebra, and computer graphics, including geometric modeling and computer animation. Some familiarity with differential geometry, differential equations, numerical computing and image processing is strongly recommended, but not required.

Speaker Biographies

David Breen is the Assistant Director of the Computer Graphics Laboratory at the California Institute of Technology. He has held research positions at the European Computer-Industry Research Centre, the Fraunhofer Institute for Computer Graphics, and the Rensselaer Design Research Center (formerly the RPI Center for Interactive Computer Graphics). Breen received a B.A. degree in Physics from Colgate University in 1982. He received M.S. and Ph.D. degrees in Computer and Systems Engineering from Rensselaer Polytechnic Institute in 1985 and 1993. Breen recently co-chaired the IEEE Symposium on Parallel and Large Data Visualization and Graphics. He has published numerous papers on a diverse set of graphics-related topics, e.g. geometric modeling,

object-oriented animation, cloth simulation, augmented reality, volume graphics and level set modeling. He is the co-editor of the book *Cloth Modeling and Animation* (AK Peters). E-mail: david@gg.caltech.edu

Ron Fedkiw is an Assistant Professor in the Computer Science Department of Stanford University. He received a Ph.D. in Applied Mathematics from the University of California, Los Angeles, and held PostDoc positions in both the UCLA Mathematics Department and the Caltech Aeronautics Department before coming to Stanford. His work at UCLA and Caltech focused on the design of new algorithms for a large variety of application areas mostly related to computational fluid dynamics. Sponsored by the Office of Naval Research at UCLA and the Department of Energy at Caltech, this work led to strong collaborative relationships with a number of scientists at the national laboratories including Los Alamos National Laboratory and Lawrence Livermore National Laboratory. Additionally, Fedkiw has been active in both computer graphics and image processing through his consulting work with Arete Entertainment, Centropolis FX, and Industrial Light & Magic.
E-mail: fedkiw@cs.stanford.edu

Stanley Osher has been a Professor of Mathematics at UCLA since 1977. He is co-inventor of the Level Set Method, TV based image restoration and high resolution nonoscillatory methods (ENO, WENO) which are widely used to compute solutions of these and other real world problems. He works in the area of scientific computing with applications to many areas, including image processing, computer vision and graphics. From 1988-95 he was cofounder and co-CEO of Cognitech, Inc. Since 1998 he has been President of Level Set Systems, Inc. He has been both a Fulbright and an Alfred P. Sloan Fellow. He received a NASA Public Service Award and was an invited speaker at the International Congress of Mathematicians. His work has been written up in the media numerous times, recently in Science News (April 1999) and Die Zeit (September 1999). Osher and Fedkiw have a book in press, which should be available before the SIGGRAPH Conference: S. Osher and R. P. Fedkiw, "The Level Set Method and Dynamic Implicit Surfaces", Springer-Verlag, NY, (2002).
E-mail: sjo@math.ucla.edu

Guillermo Sapiro is an Associate Professor with the Department of Electrical and Computer Engineering at the University of Minnesota. He works on differential geometry and geometric partial differential equations, both in theory and applications in computer vision, computer graphics, and medical imaging. Sapiro has recently written a book in the area (Cambridge University Press) and co-edited special issues in the IEEE Transactions on Image Processing and the Journal of Visual Communication and Image Representation. In recognition of his contributions in this area, he was awarded the Gutwirth Scholarship for Special Excellence in Graduate Studies in 1991, the Ollendorff Fellowship for Excellence in Vision and Image Understanding Work in 1992, the Rothschild Fellowship for Post-Doctoral Studies in 1993, the Office of Naval Research Young Investigator Award in 1998, the Presidential Early Career Award for Scientist and Engineers (PECASE) in 1998, and the National Science Foundation Career Award in 1999.
E-mail: guille@ece.umn.edu

Ross Whitaker is an Assistant Professor in the School of Computing at the University of Utah and a member of the Utah Institute for Scientific Computing and Imaging. He received a B.S. degree in Electrical Engineering and Computer Science from Princeton University in 1986. He received a Ph.D. in Computer Science from the University of North Carolina, Chapel Hill in 1993. His dissertation, entitled “Geometry-Limited Diffusion”, examined techniques and applications of anisotropic diffusion for 2D and 3D image processing. In 1994 he joined the European Computer-Industry Research Centre in Munich, Germany as a research scientist and studied the application of image processing to augmented reality. From 1996 to 2000 he was an Assistant Professor at the University of Tennessee, and in July 2000 he joined the faculty of the University of Utah. He teaches image processing, computer vision, and pattern recognition. His research interests include computer vision, image processing, medical imaging, and computer graphics/visualization. He has published several papers on the application of PDE and level set methods to image processing and computer graphics since 1993.
E-mail: whitaker@cs.utah.edu

Web Sites

Osher Home Page
http://www.math.ucla.edu/~sjo

UCLA CAM Technical Reports
http://www.math.ucla.edu/applied/cam

Level Set Systems, Inc.
http://www.levelset.com

Sapiro Book
http://www.ece.umn.edu/users/guille/book.html

Mauch - Distance Transform
http://www.acm.caltech.edu/~seanm/software/cpt/cpt.html
http://www.acm.caltech.edu/~seanm/doc/cpt/html

VISPack Web Site
http://www.cs.utah.edu/~whitaker/vispack

Fedkiw Home Page
http://www.graphics.stanford.edu/~fedkiw

Breen - Geometric Modeling and Deformable Models
http://www.gg.caltech.edu/~david/david_geom_mod.html
http://www.gg.caltech.edu/~david/david_deform_mod.html

Sethian Home Page
http://www.math.berkeley.edu/~sethian

Citations For Included Papers

- M. Bertalmio, A. Bertozzi and G. Sapiro, "Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting" *Proc. Conference on Computer Vision and Pattern Recognition*, pp. 730-764, December 2001.
- D.E. Breen, S. Mauch and R.T. Whitaker, "3D Scan Conversion of CSG Models into Distance Volumes," *Proc. 1998 Symposium on Volume Visualization*, pp. 7-14, October 1998.
- D.E. Breen and R.T. Whitaker, "A Level-Set Approach for the Metamorphosis of Solid Models," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 7, No. 2, pp. 173-192, April-June 2001.

- D. Enright, S. Marschner and R. Fedkiw, "Animation and Rendering of Complex Water Surfaces," *Proc. SIGGRAPH 2002 Conference*, July 2002.
- N. Foster and R. Fedkiw, "Practical Animation of Liquids," *Proc.. SIGGRAPH 2001 Conference*, July 2001.
- F. Memoli and G. Sapiro, "Fast Computation of Weighted Distance Functions and Geodesics on Implicit Hyper-Surfaces," *Journal of Computational Physics*, Vol. 173, pp. 730-764, November 2001.
- K. Museth, D.E. Breen, R.T. Whitaker and A.H. Barr, "Level Set Surface Editing Operators," *Proc. SIGGRAPH 2002 Conference*, July 2002.
- D.Q. Nguyen, R. Fedkiw and H.W. Jensen, "Physically Based Modeling and Animation of Fire," *Proc. SIGGRAPH 2002 Conference*, July 2002.
- S. Osher and R. Fedkiw, "Level Set Methods: An Overview and Some Recent Results," *Journal of Computational Physics*, Vol. 169, pp. 475-502, 2001.
- R.T. Whitaker, "A Level-Set Approach to 3D Reconstruction From Range Data," *International Journal of Computer Vision* , 29(3), pp. 203-231), October 1998.
- R.T. Whitaker, "Reducing Aliasing Artifacts in Binary Volumes," *Proc. IEEE Symposium on Volume Visualization and Graphics*, pp. 23-32, October 2000.
- H.-K. Zhao, S. Osher and R. Fedkiw, "Fast Surface Reconstruction Using the Level Set Method," *Proc. 1st IEEE Workshop on Variational and Level Set Methods*, pp. 194–202, 2001.

Course Schedule

Module 1 - Introduction to PDE and Level Set Methods
- 8:30 Welcome - Breen
- 8:40 Introduction to PDEs and Solution Methods - Sapiro
- 9:30 Introduction to Level Set Methods and Technology - Osher

10:15 Break

Module 2 - Level Set Applications I
- 10:30 Fast Surface Reconstruction Using the Level Set Method - Osher
- 10:55 Dynamic Visibility in an Implicit Framework - Osher
- 11:15 3D Scan Conversion of Geometric Models - Breen
- 11:35 Level Set 3D Model Morphing - Breen
- 11:55 Level Set Surface Editing - Breen

12:15 Lunch Break

Module 3 - PDE Applications and Implementation
- 1:30 Image Inpainting - Sapiro
- 1:45 Computing Geodesics and Generalized Geodesics for Computer Graphics - Sapiro
- 1:55 A Geodesic Framework for Segmentation - Sapiro
- 2:05 Edge Tracing in 2D/3D - Sapiro
- 2:15 Shape and Color Preserving Representation of High Dynamic Range Images - Sapiro
- 2:25 Pattern Formation in 3D - Sapiro
- 2:35 Surface Fairing - Sapiro
- 2:45 VISPACK: An Object-Oriented Library for Processing Volumes, Images, and Level Set Surface Models - Whitaker

3:15 Break

Module 4 - Level Set Applications II
- 3:30 Surface Reconstruction from Range Data - Whitaker
- 3:45 Direct Sinogram Reconstruction - Whitaker
- 3:55 Image/Shape Blending - Whitaker
- 4:05 Antialiasing Binary Volumes - Whitaker
- 4:15 Animation and Rendering of Complex Water Surfaces - Fedkiw
- 4:45 Physically Based Modeling and Animation of Fire - Fedkiw

5:15 Course Ends

Table of Contents

Fast Computation of Weighted Distance Functions and Geodesics on Implicit Hyper-Surfaces
F. Memoli and G. Sapiro
Solving PDE's on Implicit Surfaces: Good Bye Triangulated Surfaces? (Slides)
G. Sapiro
A Framework for Solving Surface Partial Differential Equations for Computer Graphics Applications
M. Bertalmio, G. Sapiro, L.-T. Cheng and S. Osher
A Method for Denoising Textured Surfaces
S. Betelu, A Tannenbaum, G. Sapiro
Solving Variational Problems and Partial Differential Equations Mapping into General Target Manifolds
F. Memoli, G. Sapiro and S. Osher
Isosurfaces and Level-Set Surface Models
R. Whitaker

Module 4 - Level Set Applications II

A Level-Set Approach to 3D Reconstruction From Range Data
R. Whitaker
Reducing Aliasing Artifacts in Iso-Surfaces of Binary Volumes
R. Whitaker
Geometric Surface Processing via Normal Maps
T. Tasdizen, R. Whitaker, P. Burchard and S. Osher
Practical Animation of Liquids
N. Foster and R. Fedkiw
Animation and Rendering of Complex Water Surfaces
D. Enright, S. Marschner and R. Fedkiw
Physically Based Modeling and Animation of Fire
D. Nguyen, R. Fedkiw and H. Jensen

Module 1

Introduction to PDE and Level Set Methods

SIGGRAPH 2002

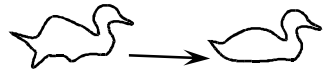
PDE's in Image Processing, Computer Vision, and Computer Graphics

Guillermo Sapiro
Electrical and Computer Engineering
University of Minnesota
guille@ece.umn.edu

Supported by NSF, ONR, NIH

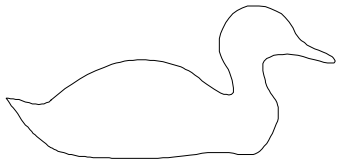
Moving Curves

Basic curve evolution



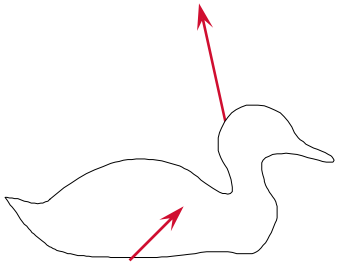
- **Planar curve:**

$$C(p) : [0,1] \rightarrow R^2$$



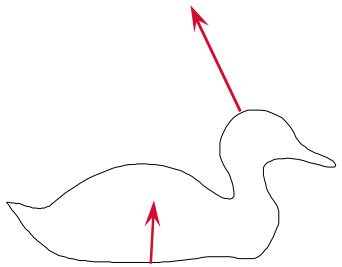
- **General flow:**

$$\frac{\partial C}{\partial t} = \alpha \vec{T} + \beta \vec{N}$$



- **General geometric flow:**

$$\frac{\partial C}{\partial t} = \beta \vec{N}$$



Mathematical morphology

- Classical theory, based on Minkowsky addition.
- The old and (probably wrong) way of doing geometric image analysis.
- Has very important lessons to learn!!!!
- Basic definitions:
  - *A*: Image in Euclidean space (*R* or *Z*)
  - *B*: Structuring element (symmetric)
  - Nothing else than Minkowsky addition

Mathematical morphology: Definitions

$$A \oplus B := \{a + b, a \in A, b \in B\} = \bigcup_{b \in B} A_b$$

$$A \ominus B := (A^c \oplus B)^c = \bigcap_{b \in B} A_b$$

$$A \circ B := (A \ominus B) \oplus B = \bigcup_{\{y : B_y \subseteq A\}} A_y$$

$$A \bullet B := (A \oplus B) \ominus B$$

Mathematical morphology: Is it good or bad?

- Advantages:
  - Nice mathematical properties (set theory)
  - Extension to Lattices
- Disadvantages:
  - Discrete Minkowsky addition does not look good, has to be replaced by better ways of computing "discrete distances."
- Major important concept: Level-sets

$$f : R^N \rightarrow R, \qquad g : R^N \rightarrow \{0, -\infty\}$$

$$f \oplus g = \max_{\text{support of } g} \{ f \}$$

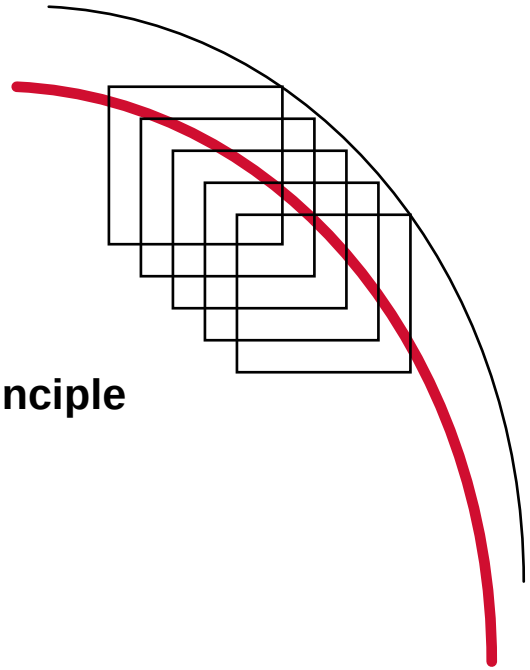
  - Commutes with thresholding (level-sets): Do binary on each level sets or do gray-level on all the image => same result
- It is in certain sense a particular case of curve evolution (before the lattices part)

Mathematical morphology via curve evolution

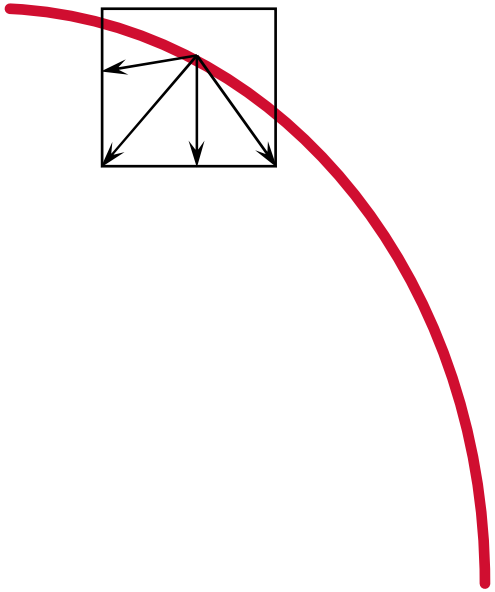
Convex structuring elements B:

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

$$A \oplus (rB) = A \oplus r_1 B \oplus r_2 B \oplus \ldots$$



Huygens principle



Mathematical morphology via curve evolution (cont.)

- General velocity:

$$\beta = \sup_{\theta}\{r(\theta)\cdot N\}$$

- Examples:

$$\beta = N \qquad B = disk$$

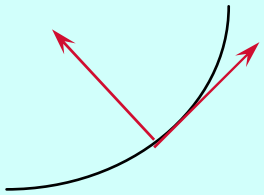
$$\beta = \max\{N_x, N_y\} \qquad B = diamond$$

$$\beta = |N_x| + |N_y| \qquad B = square$$

- Nothing else than changing the metric (distance).
- Can be explained also based on dynamic programming and time of arrival
- See Sapiro et al., Brocket-Maragos, Alvarez et al., Evans, Falcone

Planar differential geometry

- **Euclidean invariant parametrization**



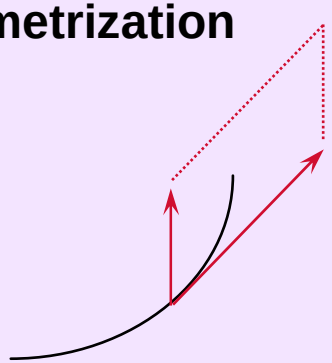
$$\left\| \frac{\partial C(s)}{\partial s} \right\| = 1$$

$$< C_s, C_{ss} > \; = \; 0$$

$$C_s \perp C_{ss}$$

$$\kappa := \; \| C_{ss} \|$$

- **Affine invariant parametrization**



$$\left\| \frac{\partial C(s)}{\partial s} \; , \; \frac{\partial^2 C(s)}{\partial s^2} \right\| = 1$$

$$\| C_s \; , \; C_{sss} \| \; = \; 0$$

$$\kappa \, C_s \; + C_{sss} \; = \; 0$$

$$\kappa \; = \; \| C_{ss} \; , \; C_{sss} \|$$

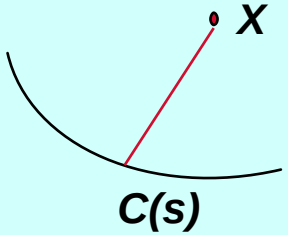
Planar differential geometry (cont.)

- Curvature constant for circles or straight lines (=0)
- Curvature defines curve up to Euclidean motion
- At least 4 points with *dk/ds=0*
- Defined for all curves

- Curvature constant for ellipses (>0), hyperbolas (<0), and parabolas (=0)
- Curvature defines curve up to affine motion
- At least 6 points with *dk/ds=0*
- Defined only for convex curves: segment at inflection points

Planar differential geometry (cont.)

$$d(X, C(s)) := \langle X - C(s)\ ,\ X - C(s) \rangle$$

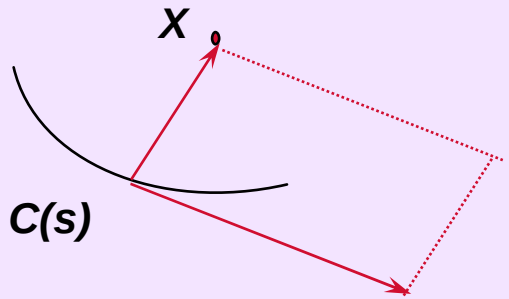


$$\frac{\partial d}{\partial s} = \langle X - C\ ,\ C_s \rangle$$

$$\Downarrow\ (=0)$$

$$X - C(s) \perp C_s(s)$$
$$X - C(s) \parallel C_{ss}(s)$$

$$d(X, C(s)) := \left\| X - C(s)\ ,\ C_s(s) \right\|$$



$$\frac{\partial d}{\partial s} = \left\| X - C(s)\ ,\ C_{ss}(s) \right\|$$

$$\Downarrow\ (=0)$$

$$X - C(s) \parallel C_{ss}(s)$$

Distance has a local extrema iff X is on the norma

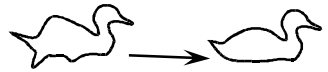
3D Differential geometry

- Remember mean and Gaussian curvatures?
- Each regular surface has two principal curvatures. The average is the mean curvature, the product the Gaussian. These are also related to the tangential map, etc, etc. See DoCarmo for details.

Riemannian geometry, Lie theory

- What about other non-Euclidean metrics?
- What about invariants to other (Lie) groups, e.g., projective?
- What about differential invariants? Semi-differential invariants? Are there any general theories?

Smoothing by classical heat flow

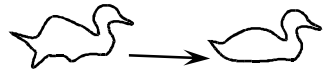


$$\frac{\partial C}{\partial t} = \Delta C \qquad \Leftrightarrow \qquad \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x_{pp} \\ y_{pp} \end{bmatrix}$$

- Linear
- Equivalent to Gaussian filtering
- Unique linear scale-space

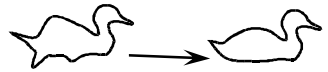
- Non geometric
- Shrinks the shape
- Implementation problems

Invariant shape deformations



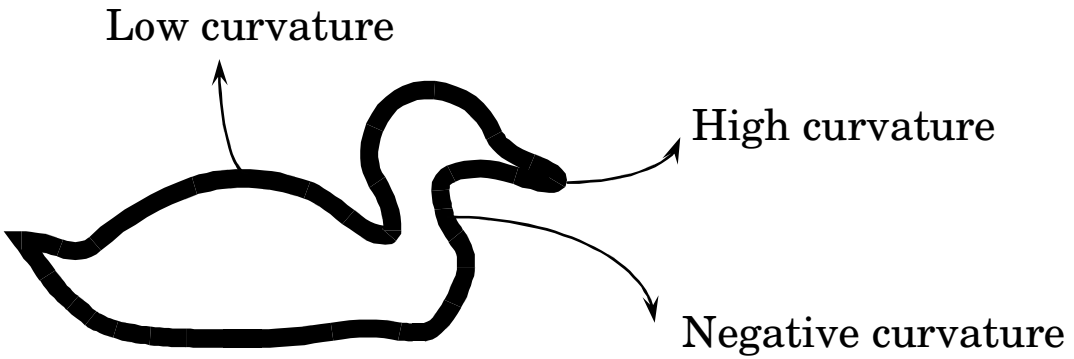
- Formulate shape deformations
  - Geometric
  - Invariant to camera transform
  - The “best” possible
  - Change only the desired features
- Motivation:
  - Mathematics:
    - From static differential geometry to dynamic
    - Beautiful
  - Computer vision and image processing:
    - Invariant shape segmentation and analysis
    - Image processing via image deformations
  - Robotics:
    - Motion planning
    - Accurate geometric object detection and tracking
    - Robot manipulation and grasping

Basic planar differential geometry

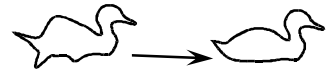


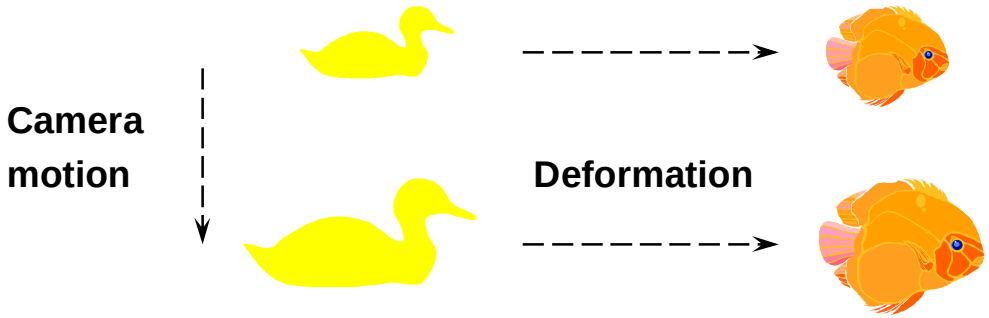
- **For every Lie group we will consider, exists and invariant parametrization s, the group *arc-length***

- **For every such a group exists and *invariant signature*, *the group curvature*, *k***

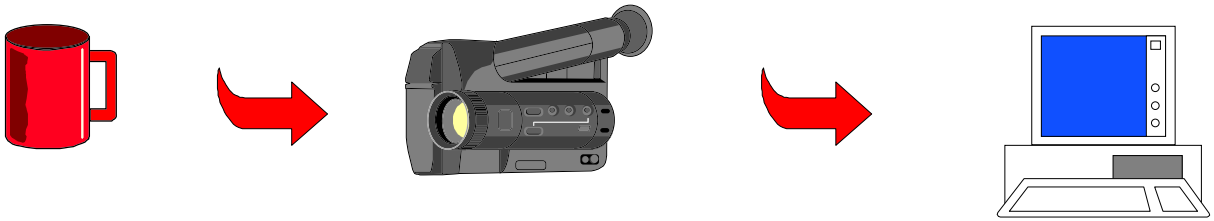


What and why invariant



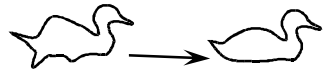


- Camera/object movement in the space



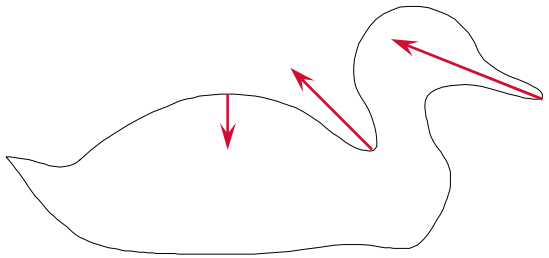
- Transformations description (for “flat” objects):
  - Euclidean
    - Motion parallel to the camera and planar projection
  - Affine
    - Planar projection
  - Projective

Euclidean geometric heat flow



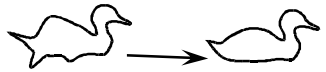
- **Use the Euclidean arc-length:** $\|C_s\| = 1$
- **The deformation:**

$$\frac{\partial C}{\partial t} = \frac{\partial^2 C}{\partial s^2} = \kappa \vec{N}$$



- **Smoothly deforms to a circle (Gage-Hamilton, Grayson)**
- **Geometric smoothing**
- **Reduces length as fast as possible**

Affine geometric heat flow (Sapiro-Tannenbaum)



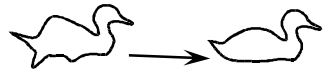
- Use the affine arc-length: $\left\| C_s \times C_{ss} \right\| = 1$
- The flow:

$$C_{ss} = \kappa^{1/3}\,\vec{N} + f(\kappa,\dot{\kappa})\,\vec{T}$$

$$\frac{\partial C}{\partial t} = \begin{cases} C_{ss} & \text{non-inflection} \\ 0 & \text{inflection} \end{cases}$$

$$C_t = \kappa^{1/3}\,\vec{N}$$

Affine geometric heat flow (cont.)



- Geometric smoothing (preserving area if desired)
  - Total curvature decreases
  - Maxima of curvature decreases
  - Number of inflections decreases
- Smoothly deforms a shape into an ellipse
- Decreases area as fast as possible (in an affine form)
- Existence also for non-smooth curves
  - Viscosity framework (Alvarez-Guichard-Morel-Lions)
  - Polygons (Angenent-Sapiro-Tannenbaum)

- Applications:
  - Curvature computation for shape recognition: reduce noise (Morel et al.)
  - Simplify curvature computation (Faugeras ‘95)
  - Object recognition for robot manipulation (Cipolla ‘95)

General invariant flows

- **Theorem:** For every sub-group of the projective group the most general invariant curve deformation has the form

$$\frac{\partial C}{\partial t} = \frac{\partial^2 C}{\partial s^2} f(\kappa, \kappa_s, \kappa_{ss}, \ldots)$$

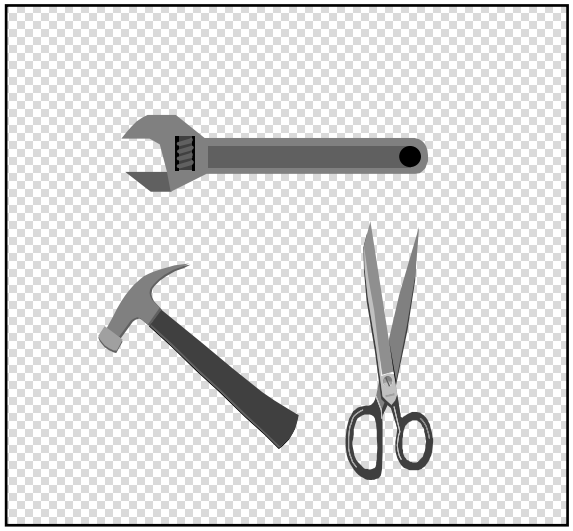
- **Theorem:** In general dimensions, the most general invariant flow is given by

$$u_t = \frac{\mathrm{g}}{E(g)} f(curvatures)$$

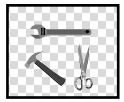
  - ***u***: graph locally representing the surface
  - ***g***: invariant metric
  - ***E(g)***: variational derivative of ***g***

- See Olver *et al.*, Alvarez *et al*., Caselles-Sbert

General Geometric Framework For Object Segmentation

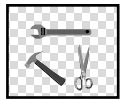


Introduction

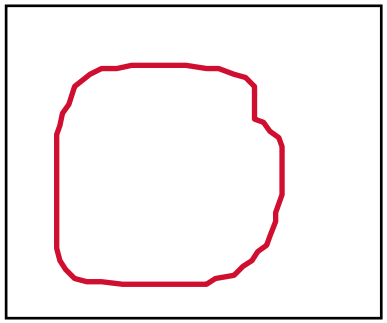


- Goal: Object detection
- Approach: Curve/surface deformation
  - Geometry dependent regularization
  - Image dependent velocity
- Characteristics:
  - Unifies previously considered independent approaches
  - Relates segmentation with anisotropic diffusion
  - General:
    - Any topology
    - Any type of image data
    - Any dimension
  - Holds formal results

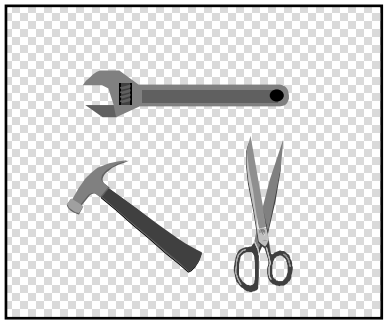
Notation



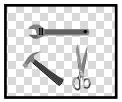
- **Deforming curve:** $C(p) : [0,1] \rightarrow R^2$



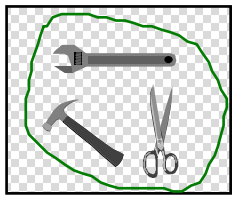
- **Image:** $I: [0,1]\times[0,1] \rightarrow R^2 \quad (R^N)$



Basic active contours approach

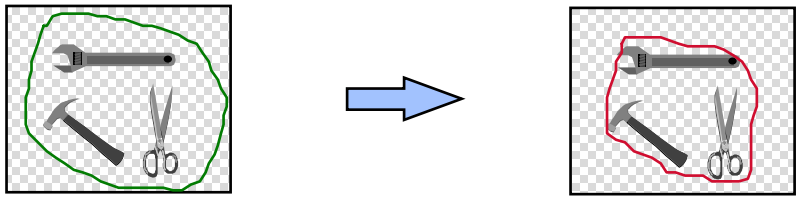


- **Terzopoulos *et al.*, Cohen *et al.***

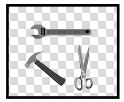


$$E(C) = \lambda \int |C'(p)|^2 \, dp + \gamma \int |C''(p)|^2 \, dp - \int |\nabla I(C)| \, dp$$

- **Drawbacks:**
  - **Too many parameters**
  - **Non-geometric**
  - **Handling topology changes**



Geodesic active contours (Caselles-Kimmel-Sapiro)



$$E(C) = \lambda \int |C'(p)|^2 \, dp + \gamma \int |C''(p)|^2 \, dp - \int |\nabla I(C)| \, dp$$

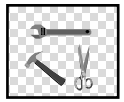
- Generalize image dependent energy
- Eliminate high order smoothness term
- Equal internal and external energies

$$E(C) = \int |C'(p)|^2 \, dp + \int g[|\nabla I(C(p))|] \, dp$$

- Maupertuis and Fermat principles of dynamical systems

$$E(C) = \int g[|\nabla I(C(s))|] \, ds$$

Geodesic computation



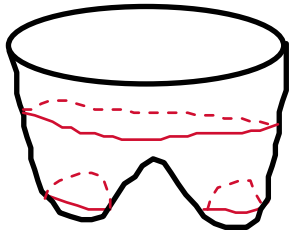
- **Gradient-descent**

$$E(C) = \int ds \quad \Rightarrow \quad \frac{\partial C}{\partial t} = \kappa \vec{N}$$

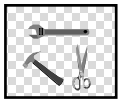
$$E(C) = \int g[|\nabla I(C(s))|]\, ds \quad \Rightarrow \quad \frac{\partial C}{\partial t} = g\,\kappa \vec{N} - \nabla g \cdot \vec{N}$$

- **Level-sets (Osher-Sethian)**

$$\frac{\partial C}{\partial t} = \beta \vec{N} \quad \Rightarrow \quad \frac{\partial \Phi}{\partial t} = \beta |\nabla \Phi|$$

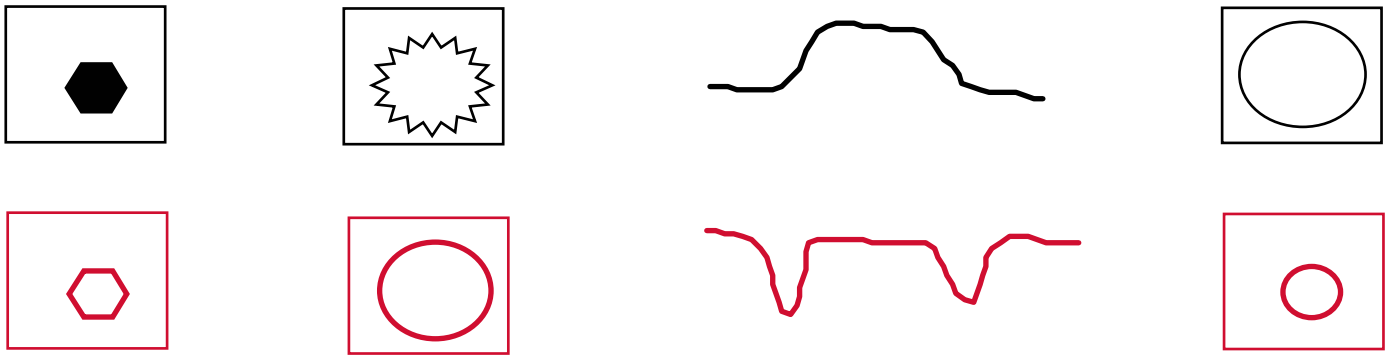


Further geometric interpretation

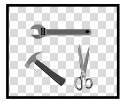


- The geodesic flow

$$\frac{\partial C}{\partial t} = g\,\kappa\vec{N} - \nabla g \cdot \vec{N} \qquad (+\,g\vec{N})$$

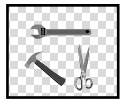


Model correctness



- **Theorem:** The deformation is independent of the level-sets embedding function
- **Theorem:** There is a unique solution to the flow in the viscosity framework
- **Theorem:** The curve converges to ideal objects when present in the image
- **Related work:**
  - Kimia-Tannenbaum-Zucker
  - Caselles *et al.*
  - Malladi-Sethian-Vemuri
  - Kichenassamy *at al.*
  - Tek-Kimia, Whitacker
- **New work:**
  - Chan-Vese
  - Paragios-Deriche
  - Yezzi et al.
  - Faugeras et al.

Extensions



$$E(C) = \int g[|\nabla I(C(s))|]\, ds$$

- **Gray-level values**
  - *ds* **- length element (geodesics)**
  - **Ordinary edge detector (gradient)**
- **Surfaces**
  - *ds* **- area element (minimal surfaces)**
  - **3D edge detector**
- **Vector-valued images (color, texture, medical, etc)**
  - *ds* **- length element**
  - **Vector-valued edge detector (vector geodesics)**
    - **Eigenvalues of the first fundamental form in Riemannian space**
- **Invariant detection (affine area geodesics)**
  - *ds* **- affine length element (area related)**
  - **Affine invariant edge detector**
  - **Affine norm for "gradient descent"**

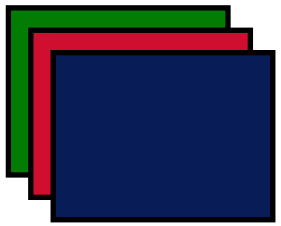
Why color edges?

Notation

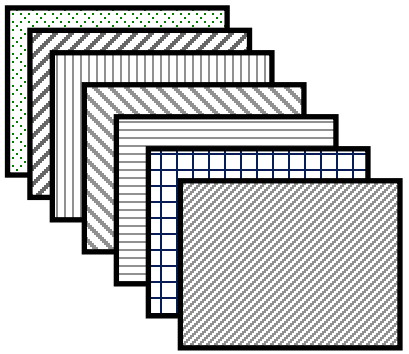
- Image

$$X : [0,N]_x[0,N] \rightarrow \mathbf{R}^N$$

$L^*a^*b^*$



- Texture: Gabor decomposition



Color edge computation

- Given a metric (Euclidean) $\Rightarrow \Delta X = \sqrt{\sum_i \Delta X_i^2}$
- Compute first fundamental form

$$[g_{ij}] = \frac{\partial X}{\partial i} \cdot \frac{\partial X}{\partial j}$$

- Compute eigenvectors and eigenvalues
- Edge: maximal eigenvalue and its eigenvector

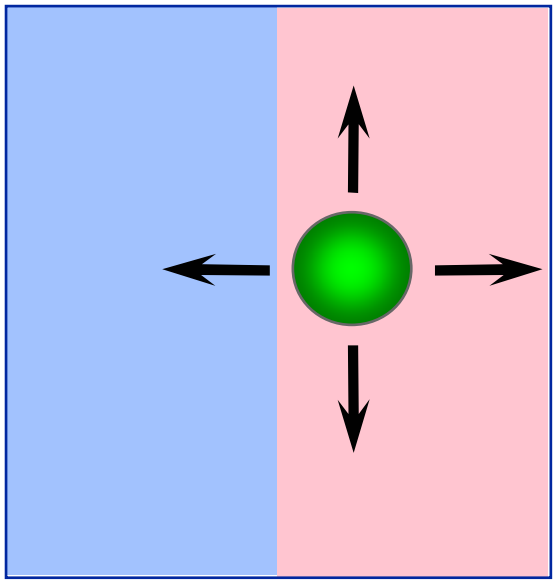
$$(\lambda_+, \lambda_-, \theta_+, \theta_-)$$

- Basic properties:
  - Eigenvectors are orthonormal
  - Minimal eigenvalue is not zero

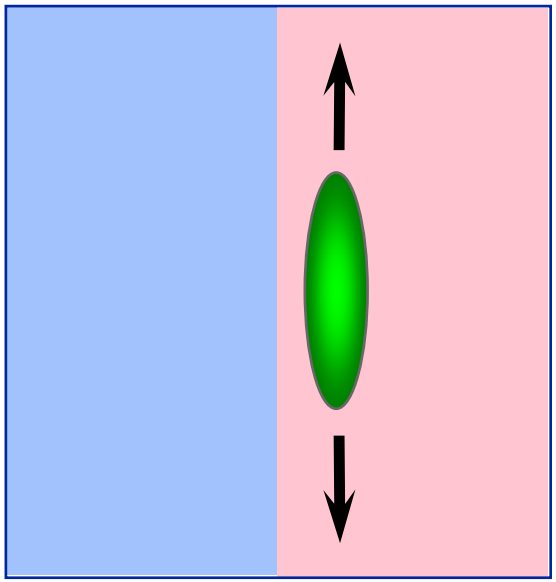
Moving Images

Anisotropic diffusion

Isotropic vs. Anisotropic Smoothing



Isotropic smoothing



Anisotropic smoothing

Isotropic diffusion (Koenderink, Witkin)

- All "equivalent:"
  - Gaussian filtering of the image Φ
  - Heat flow

$$\frac{\partial \Phi}{\partial t} = \Delta \Phi$$

  - Minimize the L2 norm

$$\int \|\nabla \Phi\|^2$$

Isotropic diffusion: Good things

- Gaussian filtering if and only if
  - Linear
  - Shift-invariant
  - No creation of zero crossings
- Gaussian filtering if and only if
  - Linear
  - Shift-invariant
  - Semi-group property
  - Scale-invariant (dimensionless)
- Unique linear filter that defines a scale-space: Do not creates information at coarser scales
- Where everything started (Koenderink, Witkin)

Isotropic diffusion: Bad things and possible solutions

- Non-geometric
- Problems with implementations

- Who said linear? Replace heat flow by “parabolic” PDE’s (Hummel’s original idea)

- Why parabolic? Because of the maximum principle.

Perona-Malik anisotropic diffusion

- Replace the L2 by a different norm (e.g., L1, Rudin-Osher-Fatemi; Lorentzian, Black et. al.; etc)

$$\iint h(\|\nabla \Phi\|) \quad \Rightarrow \quad \frac{\partial \Phi}{\partial t} \quad = \quad \mathrm{div}\left(h'(I) \, \frac{\nabla \mathrm{I}}{\|\nabla I\|} \right)$$

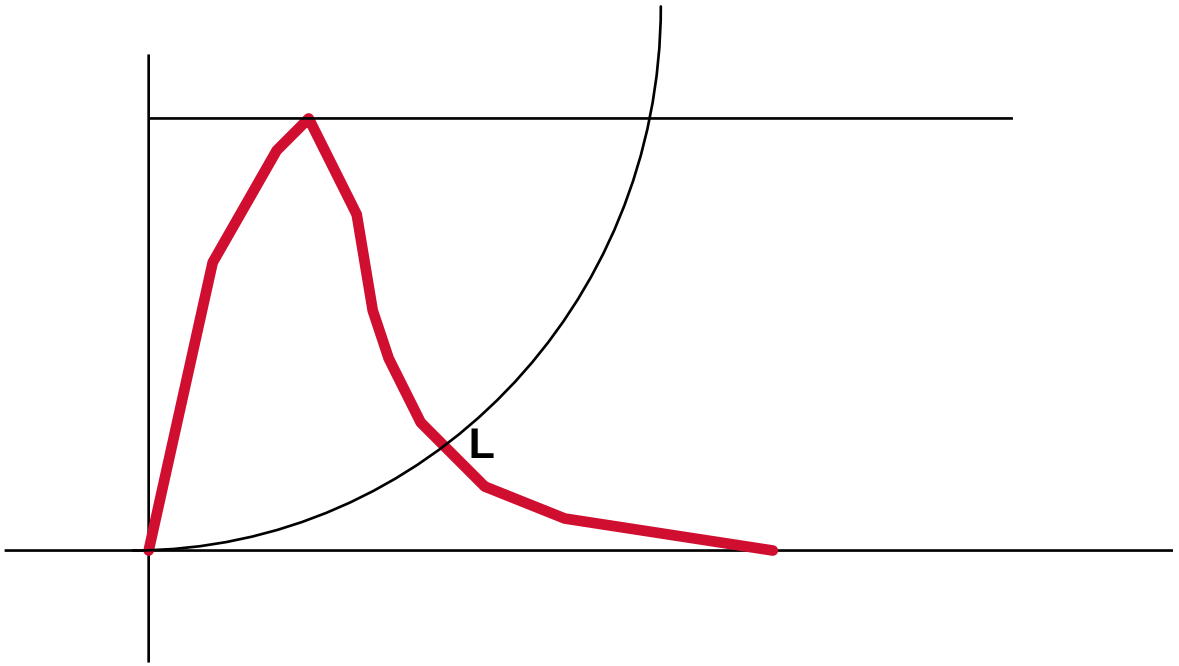
Selection of “stopping term” h

- How do we select h?
- h=x*x => L2 => linear => Isotropic diffusion
- h=x => L1 (Rudin-Osher-Fatemi)

$$\iint\left(\left\|\nabla\Phi\right\|\right) \Rightarrow \frac{\partial\Phi}{\partial t} = \kappa$$

Robust anisotropic diffusion

- General theory for selection “h”, based on the theory of influence functions in robust statistics
- Edges should be considered outliers: At certain point, h’, the influence, should be zero.



Directional diffusion

- Diffuse in the direction perpendicular to the edges (Avarez et al.)

$$\frac{\partial \Phi}{\partial t} = \kappa \left\| \nabla \Phi \right\| = \Phi_{\zeta\zeta}$$

$$\zeta \perp \nabla \Phi$$

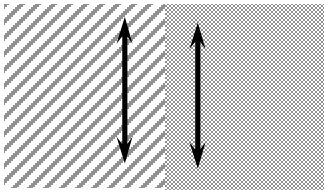
From active contours to anisotropic diffusion

- **Replace embedding function in level-sets formulation by image itself**

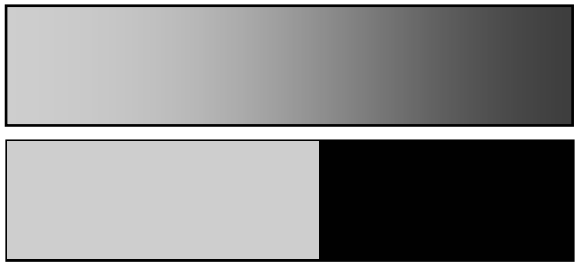
$$\frac{\partial \Phi}{\partial t} = g(I)\,\kappa\,\|\nabla\Phi\| + \nabla g\,(\mathrm{I})\cdot\nabla\Phi$$

$$\frac{\partial I}{\partial t} = g(I)\,\kappa\,\|\nabla I\| + \nabla g(I)\cdot\nabla I$$

Anisotropic diffusion (Alvarez et al.)



Shock-filters (Osher-Rudin)



Relation with Perona-Malik anisotropic diffusion

$$\frac{\partial \Phi}{\partial t} = g(I)\,\kappa\|\nabla\Phi\| + \nabla g\,(\mathrm{I})\cdot\nabla\Phi$$

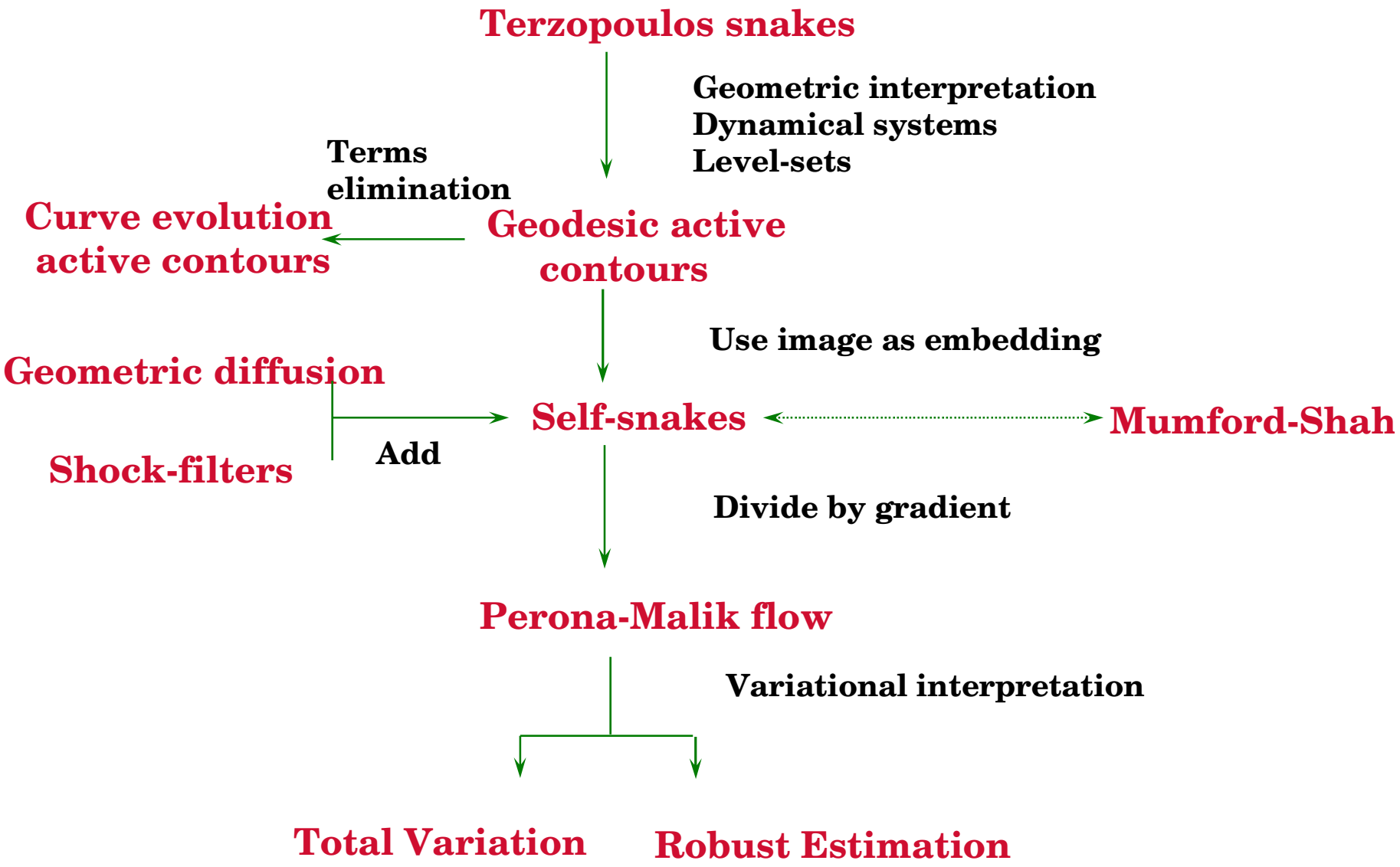
$$\Updownarrow$$

$$\frac{\partial \Phi}{\partial t} = \|\nabla\Phi\|\ \mathrm{div}\left(g(I)\,\frac{\nabla \mathrm{I}}{\|\nabla I\|}\right)$$

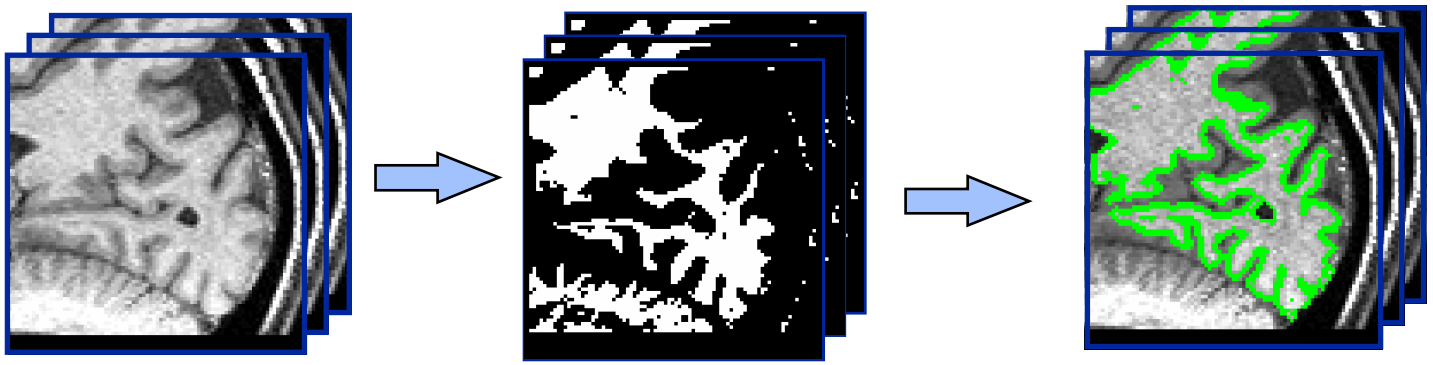
$$\iint h(\|\nabla\Phi\|) \quad \Rightarrow \quad \frac{\partial \Phi}{\partial t} = \mathrm{div}\left(h'(I)\,\frac{\nabla \mathrm{I}}{\|\nabla I\|}\right)$$

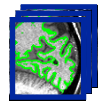
Total variation, Robust estimation Anisotropic diffusion

Concluding remarks



Anisotropic Diffusion of the Posterior





Review: MAP Estimation

- 3 classes: sulcus, gray matter, white matter
- Prior probability: Pr(class=C)
- Posterior probability: Pr(class=C | data)
- *MAP*: Choose class C that maximizes posterior:

$$C^* = \arg\max_C \Pr(class=C \mid data)$$

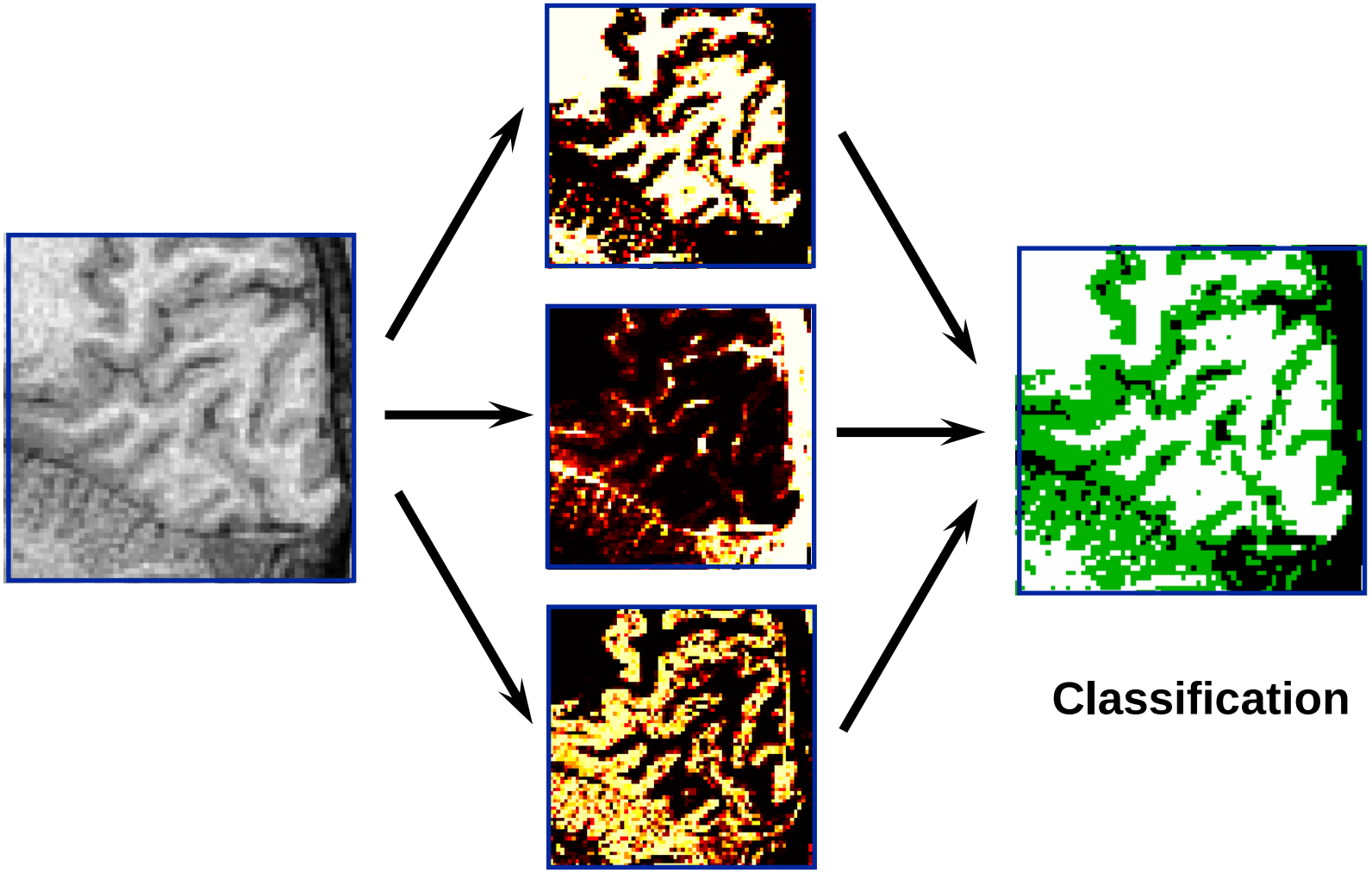
- *Bayes' Rule*:

$$\Pr(class=C \mid data) = \frac{\Pr(data \mid class=C).\Pr(class=C)}{\Pr(data)}$$

- *What is our prior,* Pr(class=C)*?*

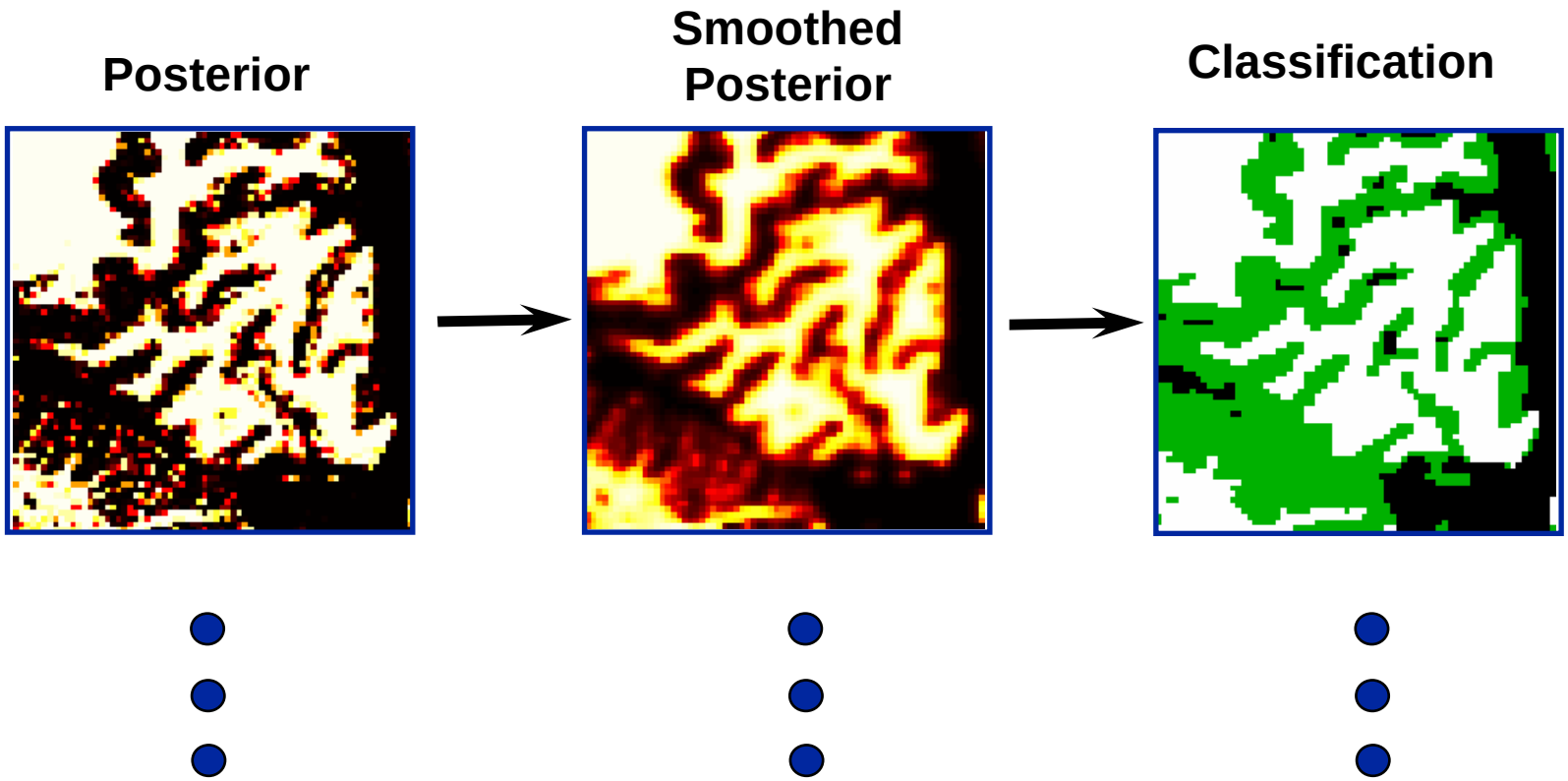
ADP: Common Techniques

MAP Estimation: Uniform Prior



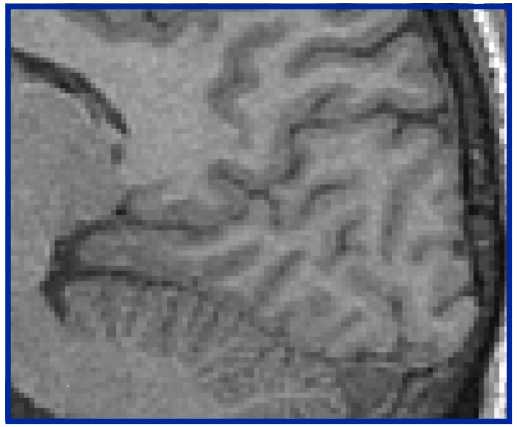
ADP: Results

Anisotropic smoothing of posterior (Teo-Sapiro-Wandell)

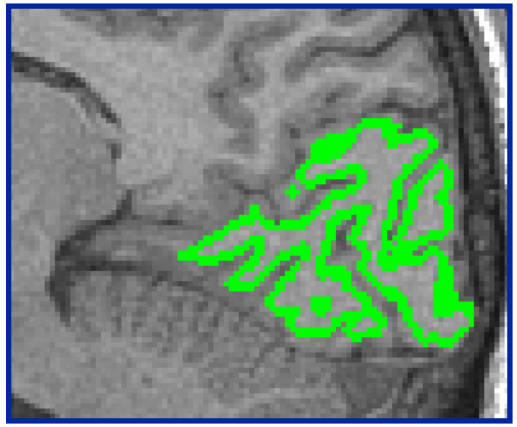


ADP: Comparisons

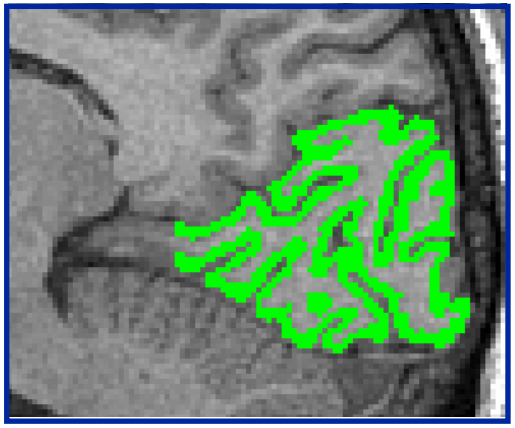
Comparison with manual segmentation



MR Image

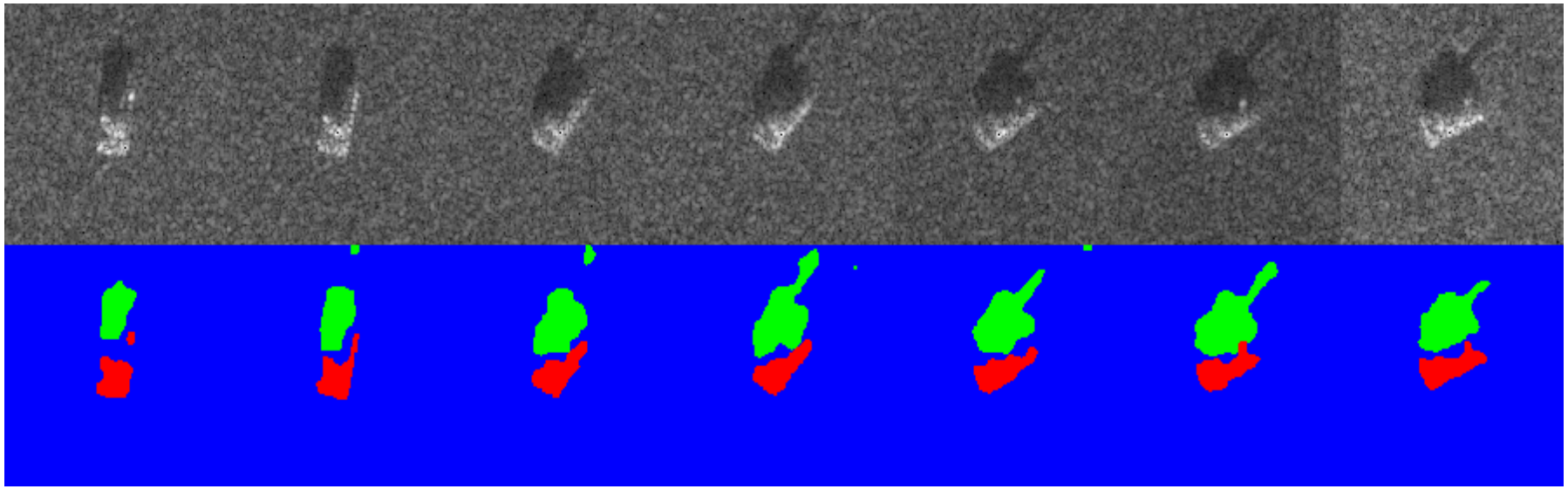


Automatic
(2 min)



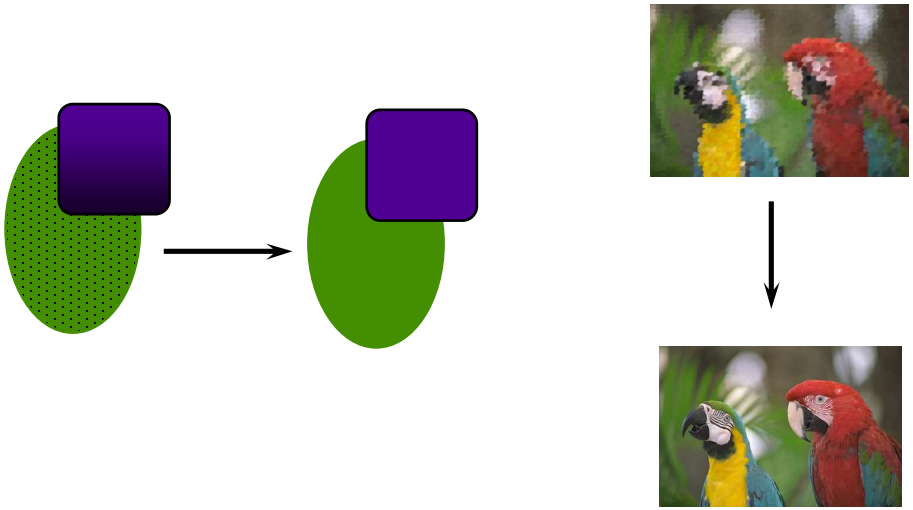
Manual
(18 hours)

SAR segmentation via vector probability processing



With A. Pardo (see also Haker-Sapiro-Tannenbaum)

Anisotropic Diffusion in Vector Space



Goal and approach (Ringach-Sapiro)

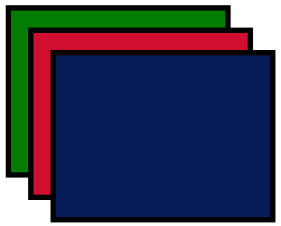
- **Goal:**
  - Enhancement of vector valued data
  - Extend classical theories of scalar PDE's in image processing
- **Approach:**
  - Work in vector space
  - Compute vector edges
  - Anisotropic diffusion
- **Important:** Works for any vector data
- **See also:** Cumani, Di Zenzo, Chambolle

Notation

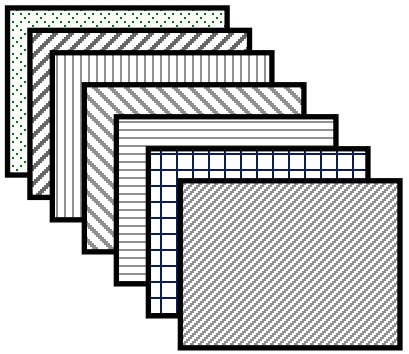
- Image

$$X : [0,N]_x[0,N] \rightarrow \mathbf{R}^N$$

$L^*a^*b^*$



- Texture: Gabor decomposition



Color edge computation

- Given a metric (Euclidean) $\Rightarrow \Delta X = \sqrt{\sum_i \Delta X_i^2}$
- Compute first fundamental form

$$[g_{ij}] = \frac{\partial X}{\partial i} \cdot \frac{\partial X}{\partial j}$$

- Compute eigenvectors and eigenvalues
- Edge: maximal eigenvalue and its eigenvector

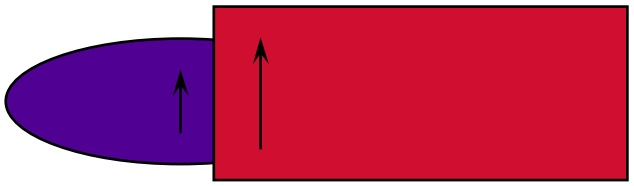
$$(\lambda_+, \lambda_-, \theta_+, \theta_-)$$

- Basic properties:
  - Eigenvectors are orthonormal
  - Minimal eigenvalue is not zero

Color anisotropic diffusion

- Direction: ***Minimal change*** (θ_{-})
- Strength: $g(\lambda_{+},\lambda_{-})$

$$\frac{\partial X}{\partial t} = g(\lambda_{+},\lambda_{-})\,\frac{\partial^2 X}{\partial \theta_{-}^2}$$

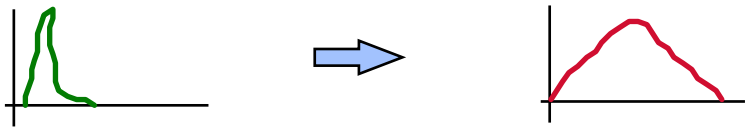


Level lines for vectorial images (Chung-Sapiro)

Vector and scalar representation
sharing level-lines

Contrast Enhancement (Sapiro-Caselles, and Caselles-Lisani-Morel-Sapiro)

- **Contrast enhancement via image deformations**
  - **Approach: Histogram modification**



$$\frac{\partial I(x,y)}{\partial t} = I(x,y) - (\#\,pixels\ of\ value \geq I(x,y)\ )$$

$$U(I) = \frac{1}{2}\int [I(\vec{x})-1/2]^2\, d\,\vec{x} - \frac{1}{4}\iint [I(\vec{x})-I(\vec{z})]\, d\,\vec{x}\, d\,\vec{z}$$

  - **Characteristics:**
    - **Simultaneous contrast enhancement and denoising**
    - **First explanation of histogram modification in image domain**
    - **Extended to local**
    - **First semi-global partial differential equation in image processing**
    - **Formal existence results**

Beyond the flat manifolds

The main problem and our goal (Tang-Sapiro-Caselles)

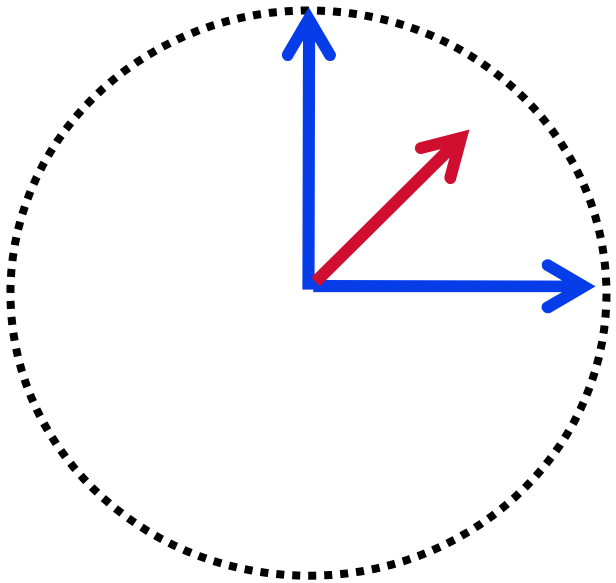
- **Goal:** Enhancement and analysis of directional data (and data on non-flat manifolds)

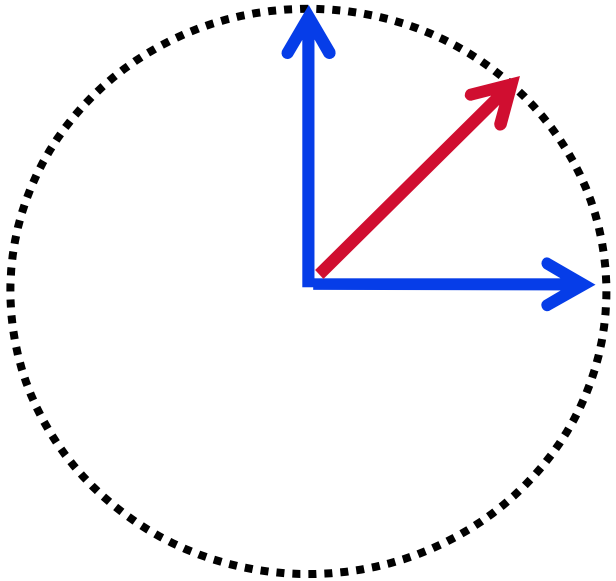
- **Problem:** Directions are unit vectors:
  - Regular images vs Directions

$$I: R^2 \rightarrow R^N \quad \text{vs} \quad I: R^2 \rightarrow S^{N-1}$$

- **Applications:**
  - Optical flow, Gradients
  - Vector data (normalized)
  - Color image enhancement
  - Surface normals and principal directions
  - Flows in general manifolds

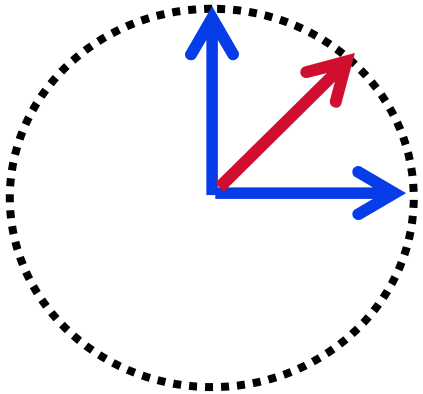
Average





Most popular previous approaches

- Work with angles: Operations on the sphere
  - Average, median, etc
  - *Statistics of directional data*, Mardia
  - “Orientation Diffusion,” Perona (1998)



- Tensor diffusion
  - Weickert, Granlund-Knuttson

- See also Chan-Shen

Anisotropic Diffusion

Isotropic
(Heat equation)

$$\frac{\partial I(x,y,t)}{\partial t} = \Delta I$$

Anisotropic

$$\frac{\partial I(x,y,t)}{\partial t} = div(g(|\nabla I|)\nabla I)$$

What have we learned from images?

Robust Estimation: $\min_I \int_\Omega \rho(|\nabla I|) d\Omega$

Robust function

$\Downarrow$

Gradient Descent: $\frac{\partial I(x,y,t)}{\partial t} = div\left(\rho' \frac{\nabla I}{|\nabla I|}\right)$

Influence function (defines outliers)

$\Downarrow$

$g := \frac{\rho'}{|\nabla I|}$

Anisotropic Diffusion: $\frac{\partial I(x,y,t)}{\partial t} = div(g(|\nabla I|)\nabla I)$

Anisotropic Diffusion

Isotropic
(Heat equation)

$$\frac{\partial I(x,y,t)}{\partial t} = \Delta I$$

Anisotropic

$$\frac{\partial I(x,y,t)}{\partial t} = div(g(|\nabla I|)\nabla I)$$

Back to Directions: Basic Idea

- Use the theory of harmonic maps
  - Find a map *I* from two manifolds *(M,g)* and *(N,h)* such that

$$\min_{I:M\to N} \int_{\Omega} \left\| \nabla_M I \right\|^p \, dvol_M$$

  - In particular, liquid crystals:

$$\min_{I:R^2\to S^{n-1}} \int_{\Omega} \left\| \nabla \; I \right\|^p \, dx\, dy$$

The Gradient-Descent Equations

Geometric ($p=2$)

$$\frac{\partial I}{\partial t} = \Delta_M I + A_N(I) < \nabla_M I, \nabla_M I >$$

Liquid crystals

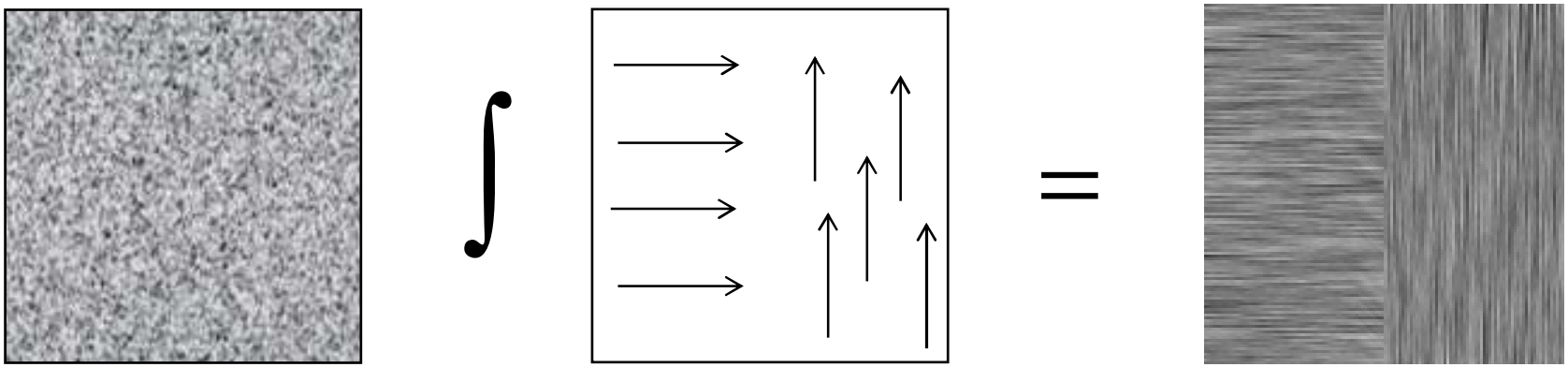
$$\frac{\partial I}{\partial t} = div(\|\nabla I\|^{p-2} \nabla I) + I\|\nabla I\|^p$$

A Few Theoretical Results (over hundreds relevant)

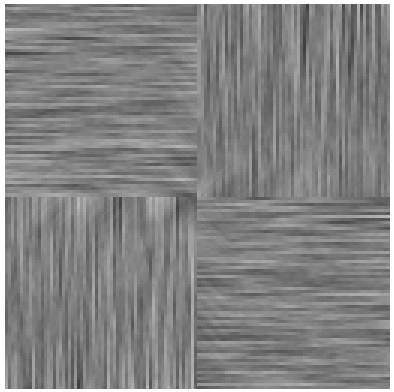
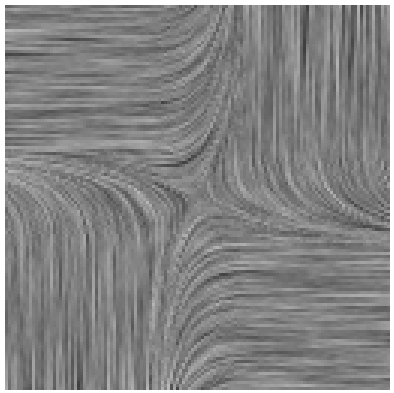
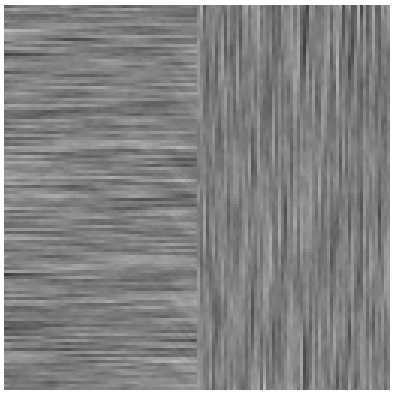
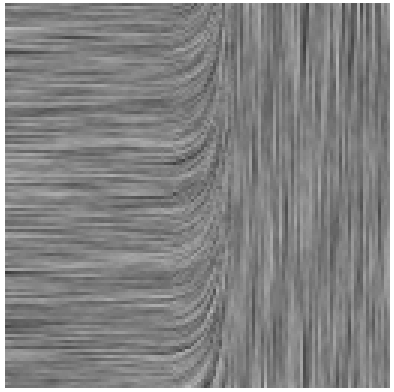
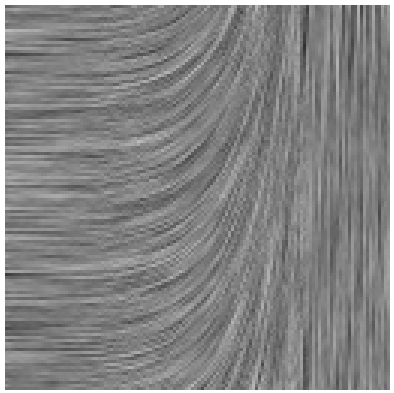
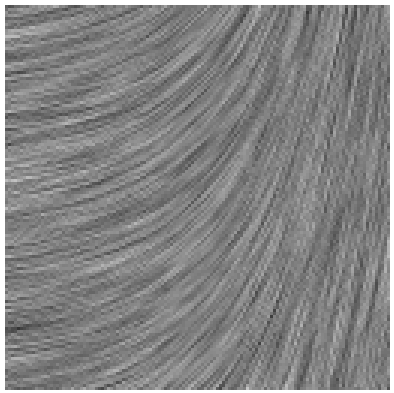
- For 2D unit vectors (n=1), and p=2, a unique solution exists and singularities are isolated points (if they exists at all). For smooth data, singularities do not occur.
- Singularities occur for 3D unit vectors (p=2).
- Singularities well characterized for 1<p<=2.
- Energy well characterized for 1<p<=2.
- No singularities for manifolds with non-positive curvature.

Intermezzo: Visualizing Directions

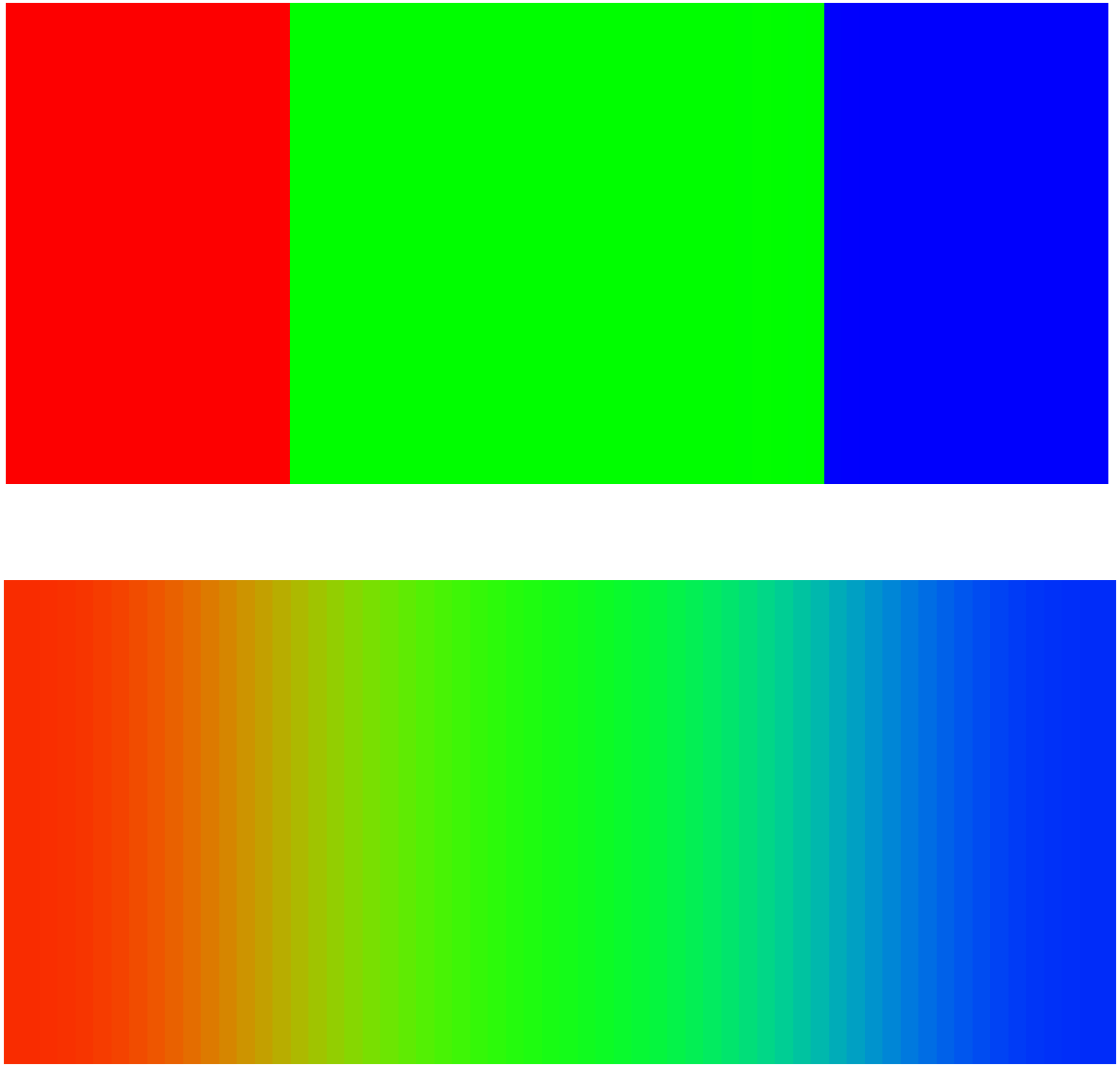
- Arrows
- Color Map
- Line Integral Convolution



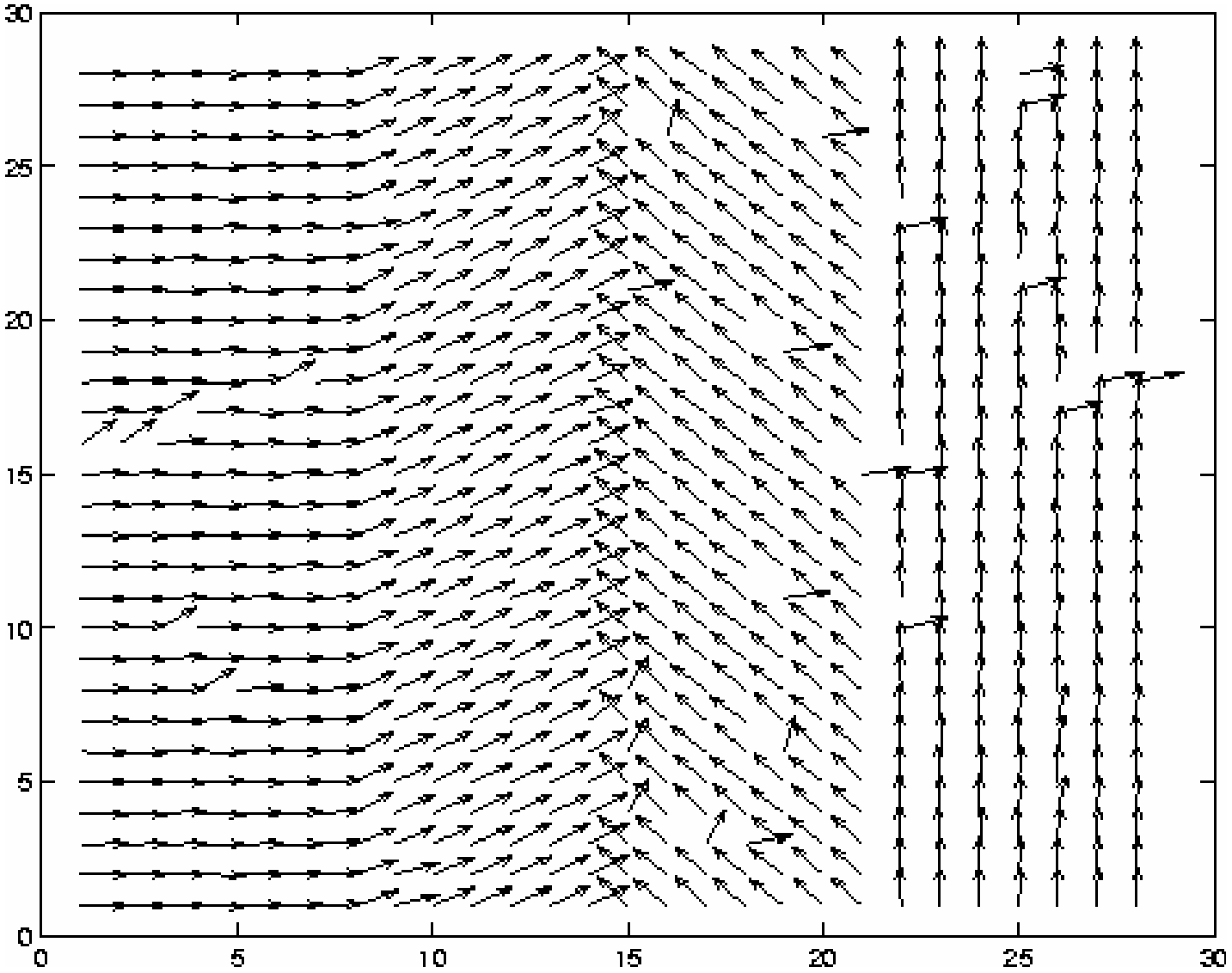
Examples (Isotropic)

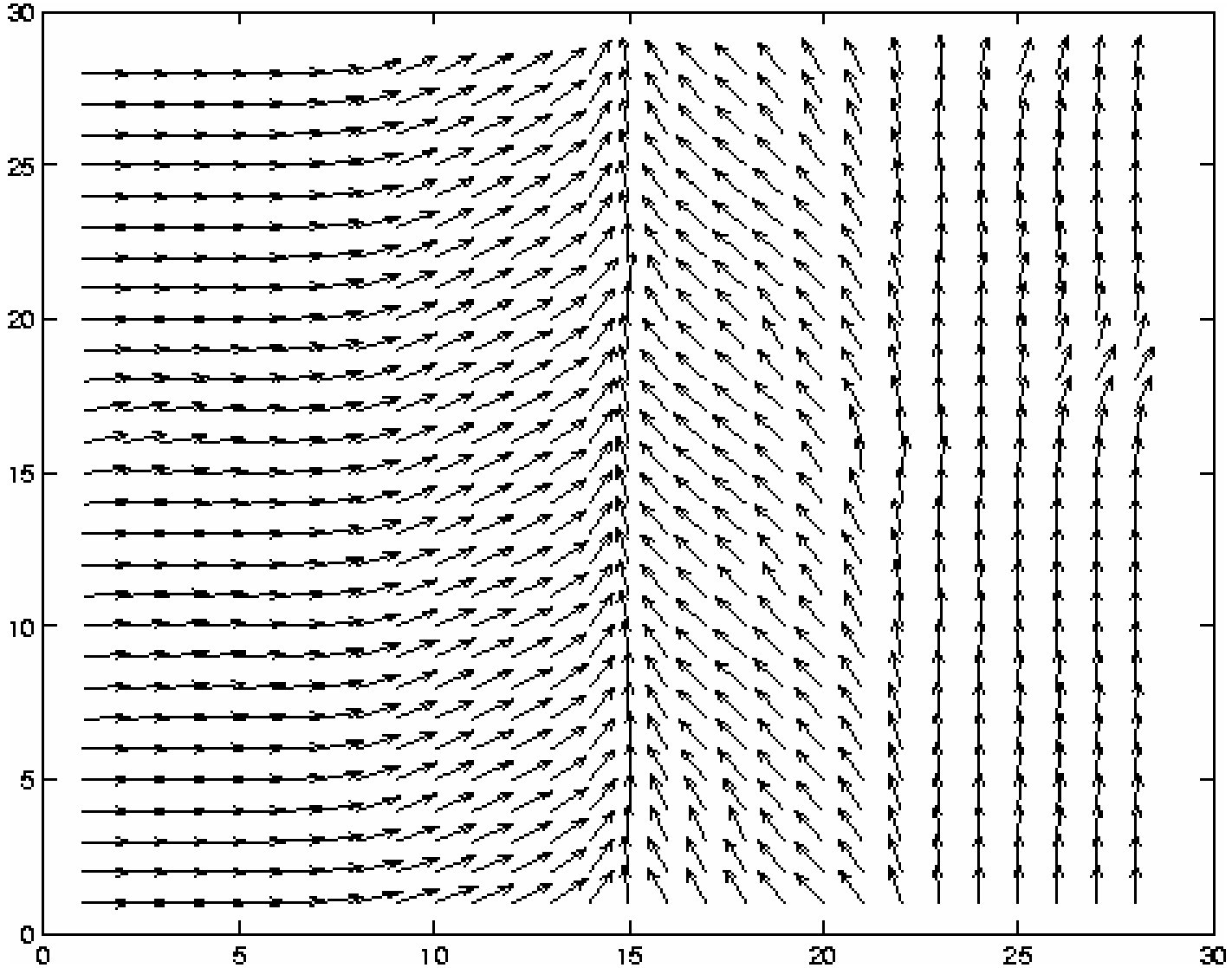







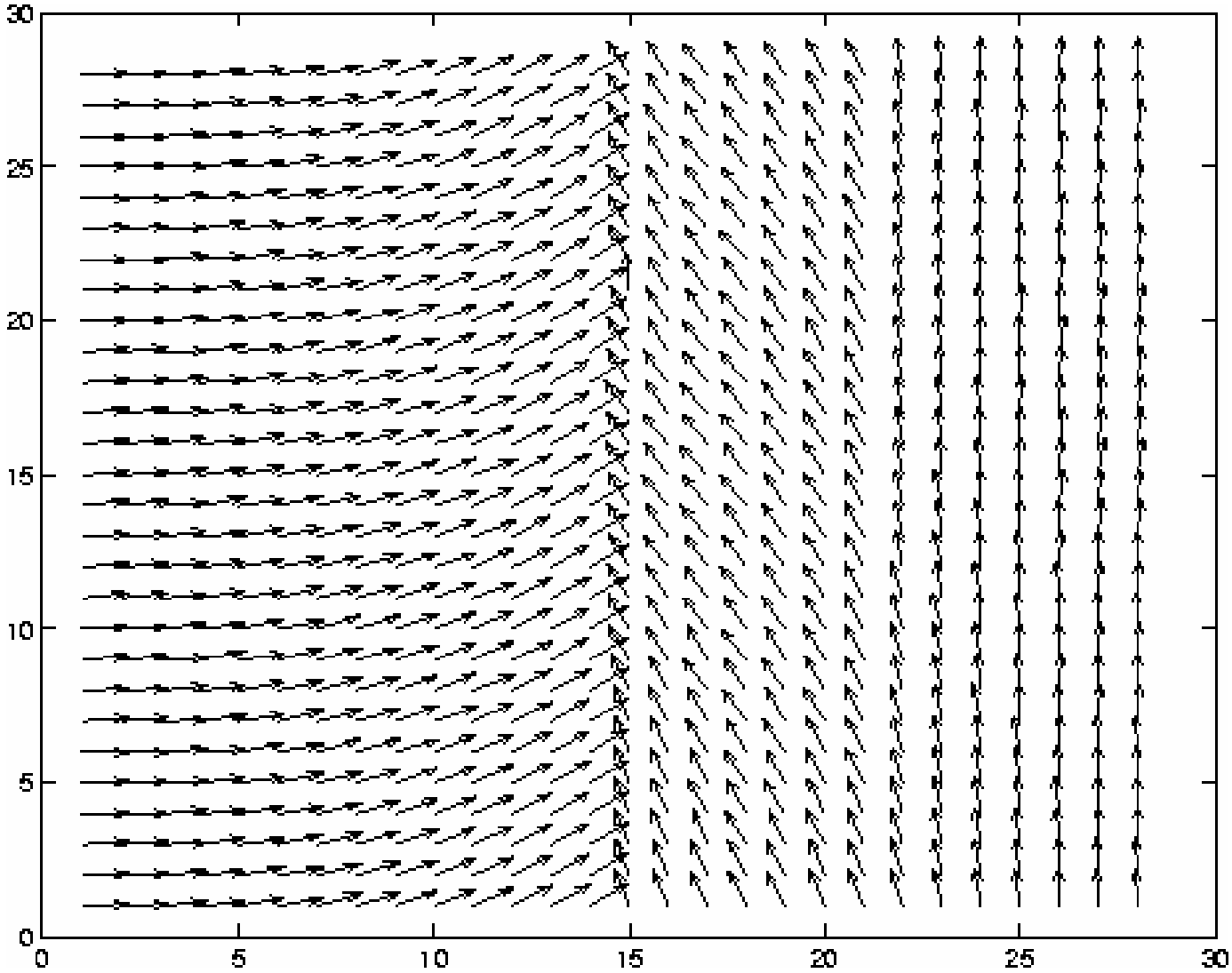
3D vector (Isotropic)



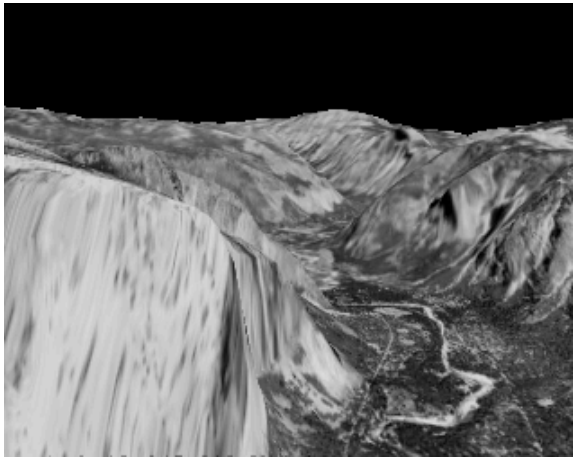
Denoising

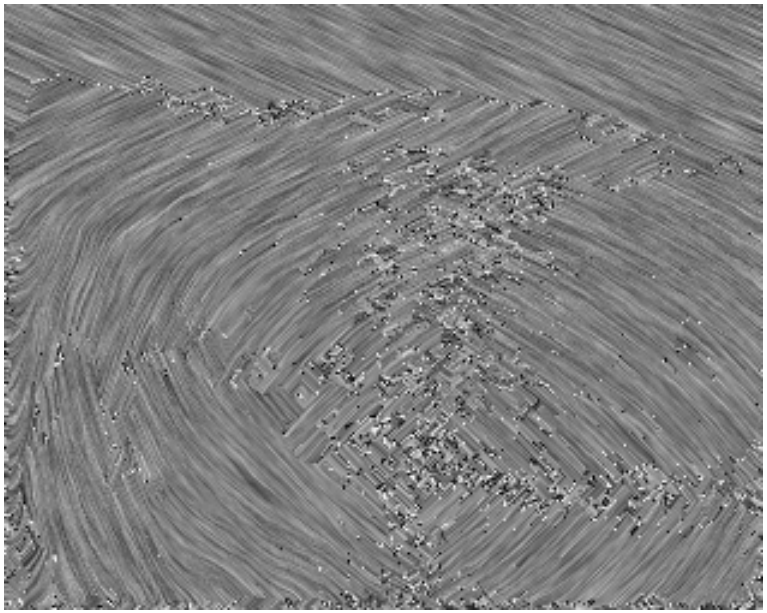


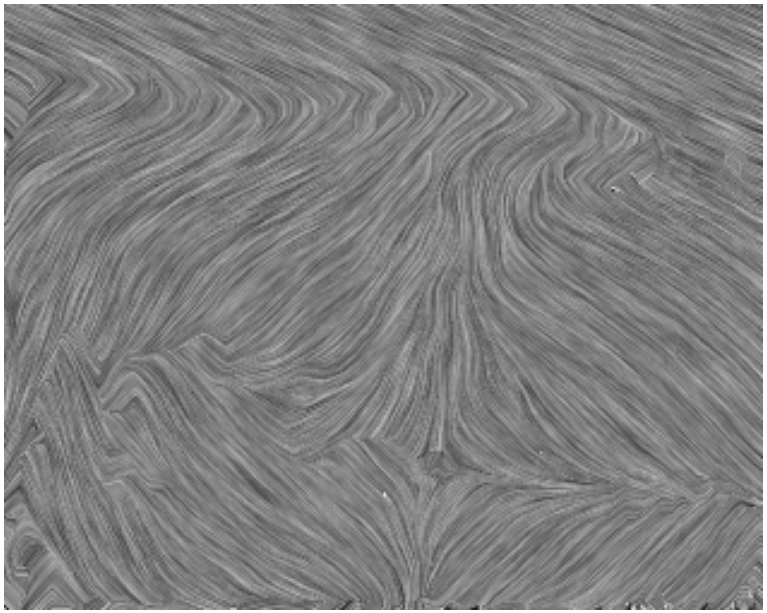

30
25
20
15
10
5
0
0
5
10
15
20
25
30


30
25
20
15
10
5
0
0
5
10
15
20
25
30

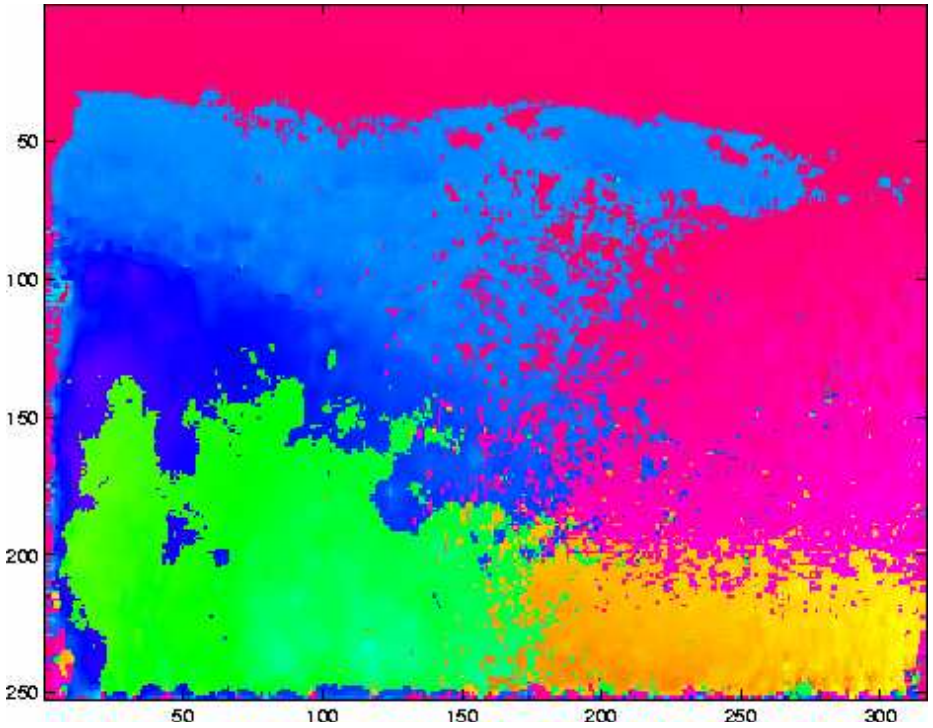
Optical flow

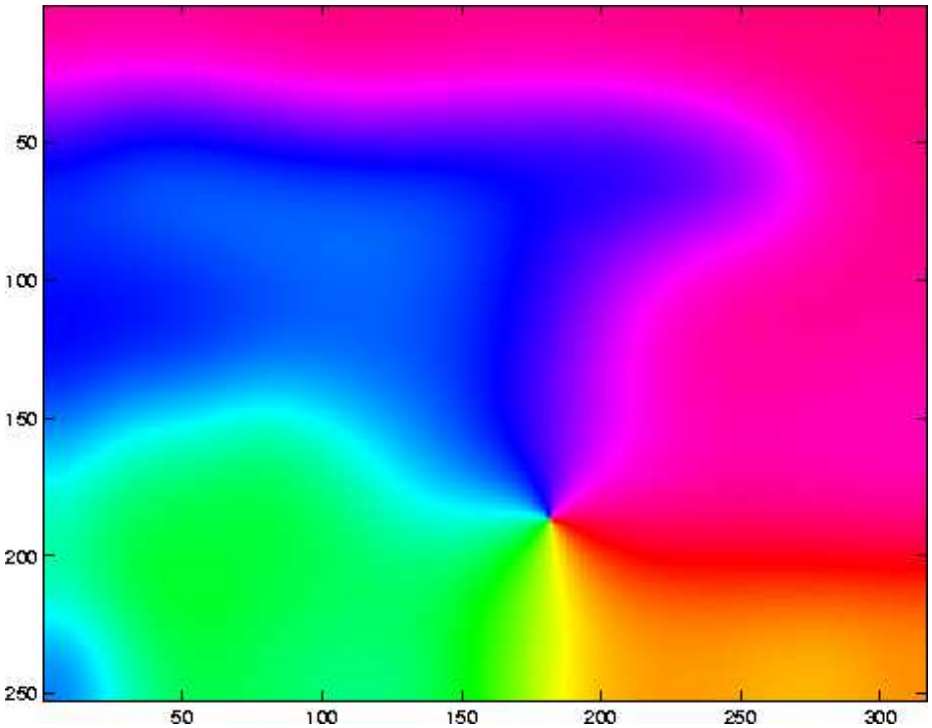




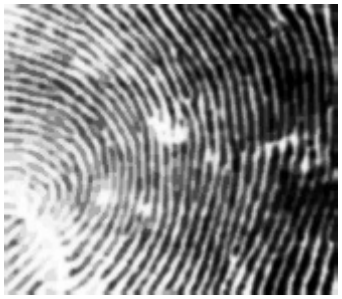


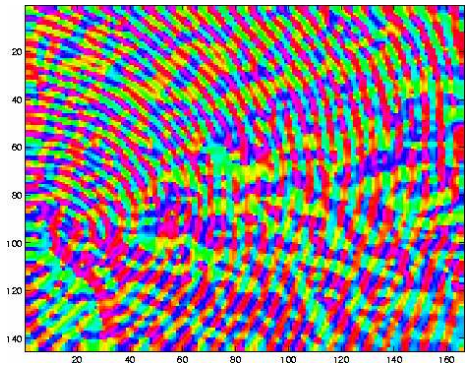
Optical flow (cont.)

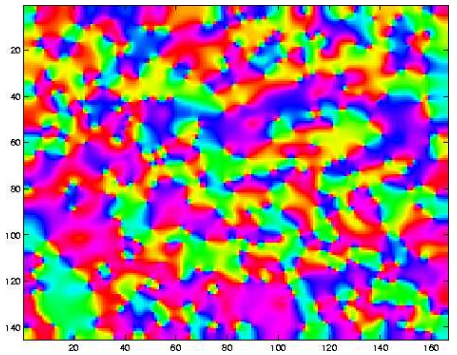


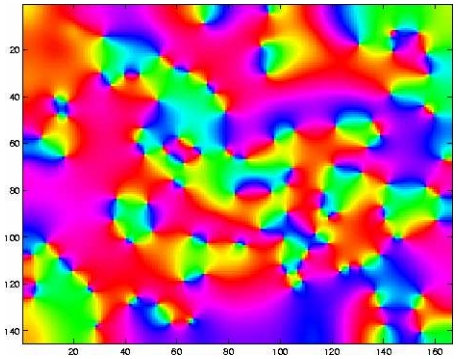


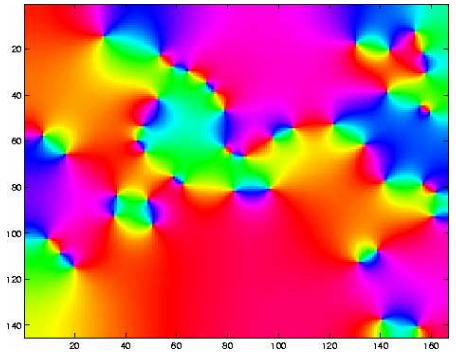
Gradient



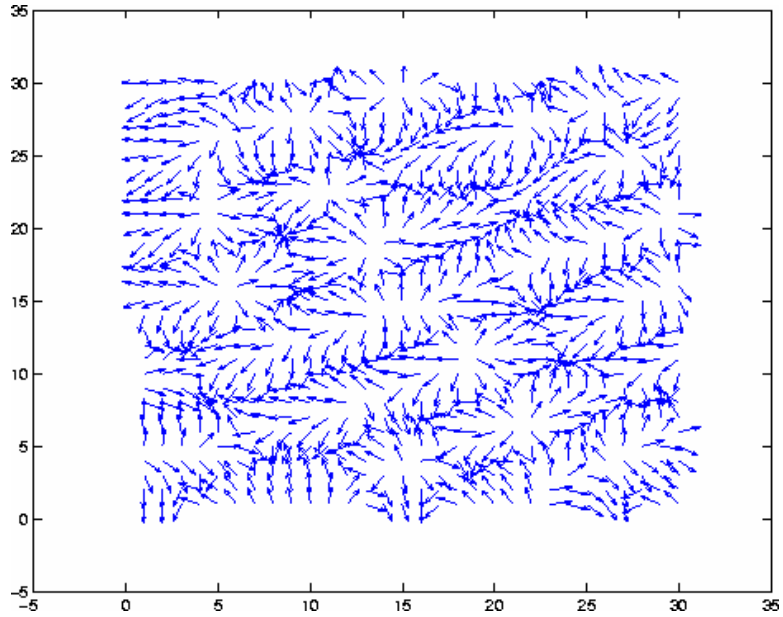

20
40
60
80
100
120
140
20
40
60
80
100
120
140
160

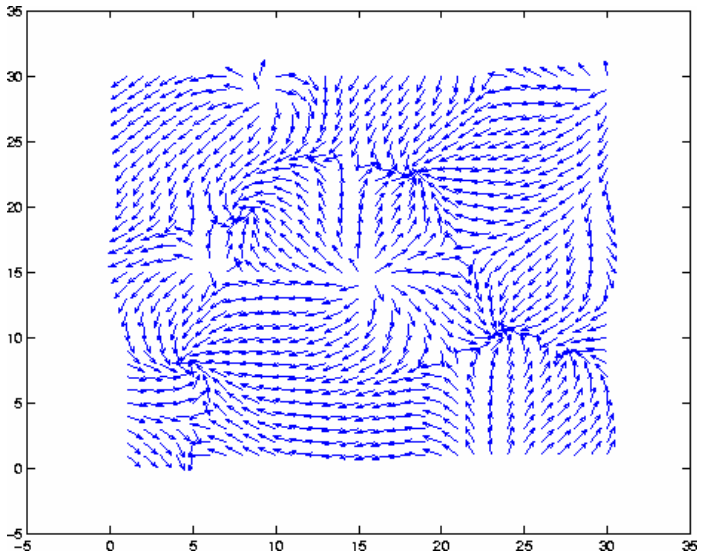

20
40
60
80
100
120
140
20
40
60
80
100
120
140
160

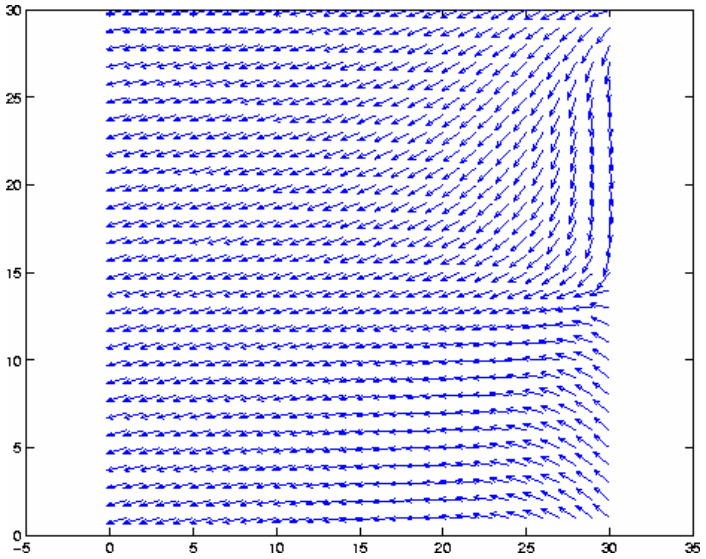

20
40
60
80
100
120
140
20
40
60
80
100
120
140
160


20
40
60
80
100
120
140
20
40
60
80
100
120
140
160

Gradient (cont.)

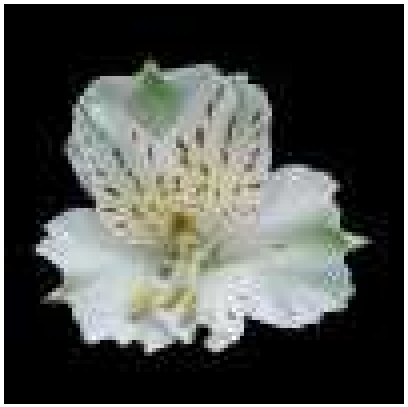






Color Image Enhancement

Color image enhancement (cont.)






Color image enhancement (cont.)

Vector probability diffusion (with Alvaro Pardo)

- Perform diffusion on the hyperplane representing probabilities

$$\min_{I:R^2 \to \mathbb{H}^n} \int_{\Omega} \left\| \nabla I \right\|^p dx\, dy$$

$$\frac{\partial I}{\partial t} = \Delta_M I + A_N(I) < \nabla_M I, \nabla_M I >$$

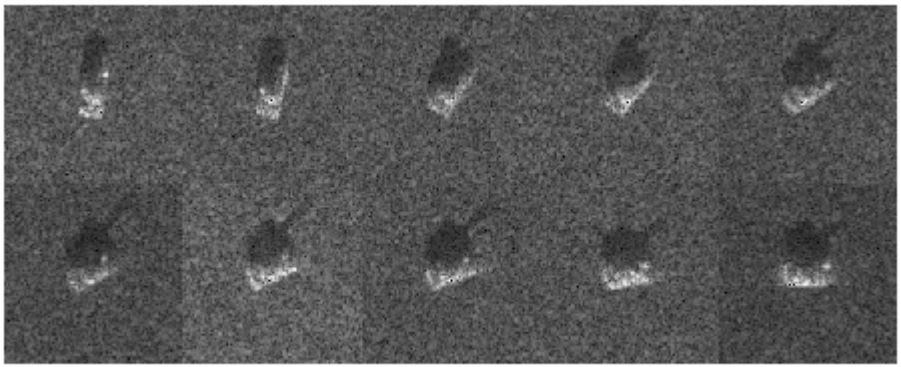
$$\frac{\partial I}{\partial t} = div(\left\| \nabla I \right\|^{p-2} \nabla I)$$

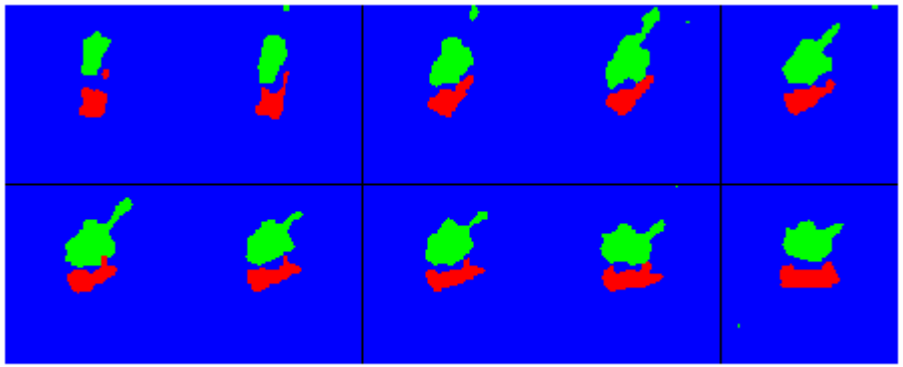
Vector probability diffusion (cont.)

- The numerical implementation also stays on the hyperplane
- The numerical implementation also holds a maximum and minimum principle

Vector probability diffusion (cont.)

- **Diffuse posterior probabilities (following Teo-Sapiro-Wandell and Haker-Sapiro-Tannenbaum)**





Current Results and Future Research

- Novel framework for analysis of directional data
  - Isotropic and anisotropic
  - Works in any dimension
  - Supported by theoretical results on existence, uniqueness, singularity classification

- See also Sochen-Kimmel-Malladi IEEE-IP ‘98, Chan-Shen UCLA ‘99, Osher-Vese UCLA ‘00.

Level Set Methods: An Overview and Some Recent Results *

Stanley Osher †
Ronald P. Fedkiw ‡

September 5, 2000

Abstract

The level set method was devised by Osher and Sethian in [64] as a simple and versatile method for computing and analyzing the motion of an interface Γ in two or three dimensions. Γ bounds a (possibly multiply connected) region Ω. The goal is to compute and analyze the subsequent motion of Γ under a velocity field $\vec{v}$. This velocity can depend on position, time, the geometry of the interface and the external physics. The interface is captured for later time as the zero level set of a smooth (at least Lipschitz continuous) function $\varphi(\vec{x}, t)$, i.e., $\Gamma(t) = \{\vec{x} | \varphi(\vec{x}, t) = 0\}$. φ is positive inside Ω, negative outside Ω and is zero on $\Gamma(t)$. Topological merging and breaking are well defined and easily performed.

In this review article we discuss recent variants and extensions, including the motion of curves in three dimensions, the Dynamic Surface Extension method, fast methods for steady state problems, diffusion generated motion and the variational level set approach. We also give a user's guide to the level set dictionary and technology, couple the method to a wide variety of problems involving external physics, such as compressible and incompressible (possibly reacting) flow, Stefan problems, kinetic crystal growth, epitaxial growth of thin films,

*Research supported in part by ONR N00014-97-1-0027, DARPA/NSF VIP grant NSF DMS9615854, AFOSR FQ8671-9801346, NSF DMS 9706827 and ARO DAAG 55-98-1-0323.

†Department of Mathematics, University of California Los Angeles, Los Angeles, California 90095

‡Computer Science Department, Stanford University, Stanford, California 94305.

vortex dominated flows and extensions to multiphase motion. We conclude with a discussion of applications to computer vision and image processing.

1 Introduction

The original idea behind the level set method was a simple one. Given an interface Γ in R^n of codimension one, bounding a (perhaps multiply connected) open region Ω, we wish to analyze and compute its subsequent motion under a velocity field $\vec{v}$. This velocity can depend on position, time, the geometry of the interface (e.g. its normal or its mean curvature) and the external physics. The idea, as devised in 1987 by S. Osher and J.A. Sethian [64] is merely to define a smooth (at least Lipschitz continuous) function $\varphi(x,t)$, that represents the interface as the set where $\varphi(x,t) = 0$. Here $x = x(x_1, \ldots, x_n)\ \varepsilon\ R^n$.

The level set function φ has the following properties

$$\begin{aligned} \varphi(x,t) &> 0 \text{ for } x \in \Omega \\ \varphi(x,t) &< 0 \text{ for } x \notin \bar{\Omega} \\ \varphi(x,t) &= 0 \text{ for } x \in \partial\Omega = \Gamma(t) \end{aligned}$$

Thus, the interface is to be captured for all later time, by merely locating the set $\Gamma(t)$ for which φ vanishes. This deceptively trivial statement is of great significance for numerical computation, primarily because topological changes such as breaking and merging are well defined and performed "without emotional involvement".

The motion is analyzed by convecting the φ values (levels) with the velocity field $\vec{v}$. This elementary equation is

$$\frac{\partial \varphi}{\partial t} + \vec{v} \cdot \nabla \varphi = 0. \tag{1}$$

Here $\vec{v}$ is the desired velocity on the interface, and is arbitrary elsewhere.

Actually, only the normal component of v is needed: $v_N = \vec{v} \cdot \frac{\nabla \varphi}{|\nabla \varphi|}$, so (1) becomes

$$\frac{\partial \varphi}{\partial t} + v_N |\nabla \varphi| = 0. \tag{2}$$

In section 3 we give simple and computationally fast prescriptions for reinitializing the function φ to be signed distance to Γ, at least near the boundary [84], smoothly extending the velocity field v_N off of the front Γ [24] and solving equation (2) only locally near the interface Γ, thus lowering the complexity of this calculation by an order of magnitude [66]. This makes the cost of level set methods competitive with boundary integral methods, in cases when the latter are applicable, e.g. see [42].

We emphasize that all this is easy to implement in the presence of boundary singularities, topological changes, and in 2 or 3 dimensions. Moreover, in the case which v_N is a function of the direction of the unit normal (as in kinetic crystal growth [62], and Uniform Density Island Dynamics [15], [36]) then equation (2) becomes the first order Hamilton-Jacobi equation

$$\frac{\partial \varphi}{\partial t} + |\nabla \varphi| \gamma \left(\vec{N} \right) = 0 \tag{3}$$

where $\gamma = \gamma(\vec{N})$ a given function of the normal, $\vec{N} = \frac{\nabla \varphi}{|\nabla \varphi|}$.

High order accurate, essentially non-oscillatory discretizations to general Hamilton-Jacobi equations including (3) were obtained in [64], see also [65] and [43].

Theoretical justification of this method for geometric based motion came through the theory of viscosity solutions for scalar time dependent partial differential equations [23], [30]. The notion of viscosity solution (see e.g. [8, 27]) – which applies to a very wide class of these equations, including those derived from geometric based motions – enables users to have confidence that their computer simulations give accurate, unique solutions. A particularly interesting result is in [29] where motion by mean curvature, as defined by Osher and Sethian in [64], is shown to be essentially the same motion as is obtained from the asymptotics in the phase field reaction diffusion equation. The motion in the level set method involves no superfluous stiffness as is required in phase field models. As was proven in [53], this stiffness due to a singular perturbation involving a small parameter ϵ will lead to incorrect answers as in [48], without the use of adaptive grids [59]. This is not an issue in the level set approach.

The outline of this paper is as follows: In section 2 we present recent variants, extensions and a rather interesting selection of related fast numerical methods. This section might be skipped at first, especially by newcomers to this subject. Section 3 contains the key definitions and basic level set technology, as well as a few words about the numerical implementation. Section 4 describes applications in which the moving interfaces are coupled to external physics. Section 5 concerns the variational level set approach with applications to multiphase (as opposed to two phase) problems. Section 6 gives a very brief introduction to the ever-increasing use of level set method and related methods in image analysis.

2 Recent Variants, Extensions and Related Fast Methods

2.1 Motion of Curves in Three Spatial Dimensions

In this section we discuss several new and related techniques and fast numerical methods for a class of Hamilton-Jacobi equations. These are all relatively recent developments and less experienced readers might skip this section at first.

As mentioned above, the level set method was originally developed for curves in R^2 and surfaces in R^3. Attempts have been made to modify it to handle objects of high codimension. Ambrosio and Soner [5] were interested in moving a curve in R^3 by curvature. They used the squared distance to the curve as the level set function, thus fixing the curve as the zero level set, and evolved the curve by solving a PDE for the level set function. The main problem with this approach is that one of the most significant advantages of level set method, the ability to easily handle merging and pinching, does not carry over. A phenomenon called "thickening" emerges, where the curve develops an interior.

Attempts have also been made in other directions, front tracking, e.g. see [41]. This is where the curve is parameterized and then numerically represented by discrete points. The problem with this approach lies in finding when merging and pinching will occur and in reparameterizing the curve when it does. The representation we derived in [13] makes use of two level set functions to model a curve in R^3, an approach Ambrosio and Soner also suggested but did not pursue because the theoretical aspects become very difficult. In this formulation, a curve is represented by the intersection between the zero level sets of two level set functions ϕ and ψ, i.e., where $\phi = \psi = 0$. From this, many properties of the curve can be derived such as the tangent vectors, $\vec{T} = \frac{\nabla\psi \times \nabla\phi}{|\nabla\psi \times \nabla\phi|}$, the curvature vectors, $\kappa\vec{N} = \nabla\vec{T} \cdot \vec{T}$, and even the torsion, $\tau\vec{N} = -\nabla\vec{B} \cdot \vec{T}$, where $\vec{N}$ and $\vec{B}$ are the normal and binormal respectively.

Motions of the curve can then be studied under the appropriate system of PDE's involving the two level set functions. The velocity can depend on external physics, as well as on the geometry of the curve (as in the standard level set approach). The resulting system of PDE's for ψ and ϕ is

$$\phi_t = -\vec{v} \cdot \nabla\phi$$
$$\psi_t = -\vec{v} \cdot \nabla\psi$$

A simple example involves moving the curve according to its curvature vectors, for which $\vec{v} = \kappa\vec{N}$. We have shown that this system can also be obtained by applying a gradient descent algorithm minimizing the length of the curve,

$$L(\phi, \psi) = \int_{R^3} |\nabla\psi \times \nabla\phi|\delta(\psi)\delta(\phi)d\vec{x}.$$

This follows the general procedure derived in [88] for the variational level set method for codimension one motion, also described in [90]. Numerical simulations performed in [13] on this system of PDE's, and shown in figures 1 and 2, show that merging and pinching off are handled automatically and follow curve shortening principles.

We repeat the observation made above that makes this sort of motion easily accessible to this vector valued level set method. Namely all geometric properties of a curve Γ which is expressed as the zero level set of the vector equation

$$\begin{aligned}\phi(x, y, z, t) &= 0\\ \psi(x, y, z, t) &= 0\end{aligned}$$

can easily be obtained numerically by computing discrete gradients and higher derivatives of the functions ϕ and ψ restricted to their common zero level set.

This method will be used to simulate the dynamics of defect lines as they arise in heteroepitaxy of non-lattice notched materials, see [79] and [80] for Lagrangian calculations.

An interesting variant of the level set method for geometry based motion was introduced in [53] as diffusion generated motion, and has now been generalized to forms known as convolution generated motion or threshold dynamics. This method splits the reaction diffusion approach into two highly simplified steps. Remarkably, a vector valued generalization of this approach, as in the vector valued level set method described above gives an alternative approach [74] to easily compute the motion (and merging) of curves moving normal to themselves in three dimensions with velocity equal to their curvature.

2.2 Dynamic Surface Extension (DSE)

Another fixed grid method for capturing the motion of self-intersecting interfaces was obtained in [73]. This is a fixed grid, interface capturing formulation based on the Dynamic Surface Extension (DSE) method of Steinhoff

et. al. [82]. The latter method was devised as an alternative to the level set method of Osher and Sethian [64] which is needed to evolve wavefronts according to geometric optics. The problem is that the wavefronts in this case are supposed to pass through each other – not merge as in the viscosity solution case. Ray-tracing can be used but the markers tend to diverge which leads to loss of resolution and aliasing.

The original (ingenious) DSE method was not well suited to certain fundamental self intersection problems such as formation of swallowtails. In [73] we extended the basic DSE scheme to handle this fundamental problem, as well as all other complex intersections.

The method is designed to track moving sets Γ of points of arbitrary (perhaps changing) codimension, moreover there is no concept of "inside" or "outside". The method is, in some sense, dual to the level set method. In the latter, the distance representation is constant tangential to a surface. In the DSE method, the closest point to a surface is constant in directions orthogonal to the surface.

The version of DSE presented in [73] can be described as follows:

For each point in R^n, set the tracked pointed $TP(\vec{x})$ equal to $CP(\vec{x})$ the closest point (to $\vec{x}$) on the initial surface Γ_0. Set $\vec{N}$ equal to the surface normal at the tracked point $TP(\vec{x})$. Let $\vec{v}(TP(\vec{x}))$ be the velocity of the tracked point.

Repeat for all steps:

(1) Evolve the tracked point $TP(\vec{x})$ according to the local dynamics $TP(\vec{x})_t = \vec{v}(TP(\vec{x}))$.

(2) Extend the surface representation by resetting each tracked point $TP(\vec{x})$ equal to the true closest point $CP(\vec{x})$ on the updated surface Γ, where Γ is defined to be the locus of all tracked points, i.e. $\Gamma = \{TP(\vec{x}) | \vec{x}\, \varepsilon\, R^n\}$.

Replace each $\vec{N}(\vec{x})$ by the normal at the updated $TP(\vec{x})$.

This method treats self intersection by letting moving sets pass through each other. This is one of its main virtues in the ray tracing case. However, it has other virtues – namely the generality of the moving set – curves can end or change dimension.

An important extension is motivated by considering first arrival times. This enables us to easily compute swallowtails, for example, and other singular points. We actually use a combination of distance and direction of

motion. One interesting choice arises when nodal values of $TP(\vec{x})$ are set equal to the "Minimizing Point"

$$MP(\vec{x}) = \min_{\vec{y}\ \varepsilon\ \text{Interface}} \beta|(\vec{x}-\vec{y})\cdot\vec{N}^{\perp}(\vec{y})| + \|\vec{x}-\vec{y}\|^2$$

for $\beta > 0$ (rather than $CP(\vec{x})$), since a good agreement with the minimal arrival time representation is found near the surface. Recall that the minimal arrival time at a point $\vec{x}$ is the shortest time it takes a ray emanating from the surface to reach $\vec{x}$. Using this idea gives a very uniform approximation and naturally treats the prototype swallowtail problem.

For the special case of curvature dependent motion we may use an elegant observation of DeGiorgi [28]. Namely the vector mean curvature for a surface of arbitrary codimension is given by $\kappa\vec{N} = -\Delta\nabla\left(\frac{d^2}{2}\right)$ where κ is the local mean curvature and d is the distance to the surface. Using the elementary, but basic fact that

$$d\nabla d = \vec{x} - CP(\vec{x})$$

where $CP(\vec{x})$ is the closest point to $\vec{x}$ on the surface, we obtain a very simple expression for vector mean curvature

$$\kappa\vec{N} = -\Delta(\vec{x} - CP(\vec{x})) = \Delta CP(\vec{x}).$$

Thus motion by a function F, of mean curvature for surfaces of arbitrary codimension can be achieved by using $\vec{v}(TP(\vec{x})) = \Delta CP(\vec{x})$. Then curvature dependent velocities are possible by using

$$\vec{v} = F(\Delta CP(\vec{x})|_{TP(\vec{x})}\cdot\vec{N})\vec{N}.$$

where numerical experiments in [73] have validated these algorithms to some degree.

A variety of interesting topics for future research is still open. In particular, adjustments need to be made if merging is desired. Moreover we can move objects with more complex topology and geometry, such as surfaces with boundaries (or curves with endpoints), objects of composite topology (such as a filament attached to a sheet) and surfaces on curves with triple point junctions (see [88], [53] and section 5 of this paper for successful level set based and diffusion generated based approaches for the codimension one case respectively).

Further work in the area of curvature dependent motions is also possible. Computationally the construction of fast extension methods and localization

as in [66] for the level set method would be of great practical importance. It would be particularly interesting to determine if surfaces fatten (or develop interiors) when mergers occur. See [9] for a detailed discussion of this phenomenon.

Additionally in [73] we successfully calculated a geometric optics expansion by retaining the wave front curvature. Thus this method has the possibility of being quite useful in electromagnetic calculations. We hope to investigate its three dimensional performance and include the effects of diffraction.

2.3 A Class of Fast Hamilton-Jacobi Solvers

Another important set of numerical algorithms involves the fast solution of steady (time independent) Hamilton-Jacobi equations. We also seek methods which are faster than the globally defined schemes originally used to solve equation 2. The level set method of Osher and Sethian [64] for time dependent problems can be localized. This means that the problem

$$\varphi_t + \vec{v} \cdot \nabla \varphi = 0$$

with $\Gamma(t) = \{\vec{x} | \varphi(\vec{x}, t) = 0\}$ as the evolving front, can be solved locally near $\Gamma(t)$. Several algorithms exist for doing this, see [66] and [2]. These both report an $O(N)$ algorithm where N is the total number of grid points on or near the front. However, the algorithm in [66] has $O(N \log(N))$ complexity because a partial differential equation based reinitialization step requires $\log(\frac{1}{\triangle x}) \approx \log(N)$ steps to converge (we are grateful to Bjorn Engquist for pointing this out). The algorithm in [2] claims $O(N)$ complexity, but this is not borne out by the numerical evidence presented there.

However for some special Hamilton-Jacobi equations there is a fast method whose formal complexity is $O(N \log(N))$, but which, in our experience, is around one order of magnitude faster than these general local methods.

The idea is as follows:

For an equation of the form

$$\tilde{H}(\vec{x}, \nabla \psi) = 0,$$

give $\psi = 0$ on a non characteristic set S:

$$\nabla \psi \cdot \tilde{H}_{\nabla \psi} \neq 0$$

then we proved in [63] that the t level set

$$\{\vec{x}|\psi(\vec{x}) = t\} = \Gamma'(t)$$

is the same as the zero level set $\Gamma(t)$ of $\varphi(\vec{x}, t)$, for $t > 0$ where φ satisfies

$$\tilde{H}\left(\vec{x}, -\frac{\nabla\varphi}{\varphi_t}\right) = 0.$$

This means that the viscosity solutions of either problem have level sets which correspond to each other. (This was also suggested in the original level set paper of Osher and Sethian [64]). Thus, one would like to find $\Gamma(t)$, the zero level set of $\varphi(x, t)$, as $\Gamma'(t)$, the t level set of $\psi(x)$.

A canonical example is the eikonal equation

$$\varphi_t + c(\vec{x})|\nabla\varphi| = 0, \quad c(\vec{x}) < 0$$

which can be replaced by:

$$|\nabla\psi| = -\frac{1}{c(\vec{x})} = a(\vec{x}) > 0.$$

So we find first arrival times instead of zero level sets.

In [86] J.N. Tsitsiklis devised a fast algorithm for the eikonal equation. He obtained the viscosity solution using ideas involving Dijkstra's algorithm, adapted to the eikonal equation, heap sort and control theory. From a numerical PDE point of view, however, Tsitsiklis had an apparently nonstandard approximation to $|\nabla\psi|$ on a uniform Cartesian grid.

In (1995) Sethian [76] and Helmsen et. al. [40] independently published what appeared to be a simpler algorithm making use of the Rouy-Tourin algorithm to approximate $|\nabla\varphi|$. This has become known as the "fast marching method". However, together with Helmsen [61] we have proven that Tsitsiklis' approximation is the usual Rouy-Tourin [69] version of Godunov's monotone upwind scheme. That is, the algorithm in [76] and [40] is simply Tsitsiklis' algorithm with a different (simpler) exposition.

Our goal here is to extend the applicability of this idea from the eikonal equation to any geometrically based Hamiltonian. By this we mean a Hamiltonian satisfying the properties:

$$H(\vec{x}, \nabla\psi) > 0, \quad \text{if } \nabla\psi \neq \vec{0} \tag{4}$$

and

$$H(\vec{x}, \nabla\psi) \text{ is homogeneous of degree one in } \nabla\psi \tag{5}$$

We wish to obtain a fast algorithm to approximate the viscosity solution of

$$\tilde{H}(\vec{x}, \nabla\psi) = H(\vec{x}, \nabla\psi) - a(\vec{x}) = 0. \tag{6}$$

The first step is to set up a monotone upwind scheme to approximate this problem. Such a scheme is based on the idea of Godunov used in the approximation of conservation laws. In Bardi and Osher [7], see also [65], the following was obtained (for simplicity we exemplify using two space dimensions and ignore the explicit $\vec{x}$ dependence in the Hamiltonian)

$$\begin{aligned} H(\psi_x, \psi_y) &\approx H^G(D^x_+\psi, D^x_-\psi_j; D^y_+\psi, D^y_-\psi) \\ &= \mathrm{ext}_{u\ \varepsilon\ I_{(u^-,u^+)}} \mathrm{ext}_{v\ \varepsilon\ I_{(v^-,v^+)}} H(u,v) \end{aligned}$$

where

$$I(a,b) = [\min(a,b), \max(a,b)]$$

$$\mathrm{ext}_u I(a,b) = \begin{cases} \min_{a\le u\le b} & \text{if } a \le b \\ \max_{b\le u\le a} & \text{if } a > b \end{cases}$$

$$u_\pm = D^x_\pm\psi_{ij} = \pm\frac{(\psi_{i\pm1,j} - \psi_{ij})}{\Delta x}, v_\pm = D^y_\pm\psi_{ij} = \pm\frac{(\psi_{i,j\pm1} - \psi_{ij})}{\Delta y}.$$

(Note, the order may be reversed in the ext operations above – we always obtain a monotone upwind scheme which is often, but not always, order invariant [65]).

This is a monotone upwind scheme which is obtained through the Godunov procedure involving Riemann problems, extended to general Hamilton-Jacobi equations [7], [65].

If we approximate

$$H(\nabla\varphi) = a(x,y)$$

by

$$H^G(D^x_+\varphi, D^x_-\varphi; D^x_+\psi; D^y_+\psi, D^y_-\psi) \tag{7}$$

for Hamiltonians satisfying (4), (5) above, then there exists a unique solution for $\psi_{i,j}$ in terms of $\psi_{i\pm1,j}, \psi_{i,j\pm1}$ and $\psi_{i,j}$. Furthermore $\psi_{i,j}$ is a nondecreasing function of all these variables.

However, the fast algorithm needs to have property $\mathcal{F}$: The solution to (7) depends on the neighboring $\psi_{\mu,\nu}$ only for $\psi_{\mu,\nu} < \psi_{i,j}$. This gives us a hint as to how to proceed.

For special Hamiltonians of the form: $H(u,v) = F(u^2, v^2)$, with F non-decreasing in these variables, then we have the following result [61]

$$H^G(u_+, u_-; v_+, v_-) = F(\max((u_+^-)^2, (u_-^+)^2); \max((v_+^-)^2, (v_-^+)^2)) \quad (8)$$

where $x^+ = \max(x, 0), x^- = \min(x, 0)$. It is easy to see that this numerical Hamiltonian has property $\mathcal{F}$ described above. This formula, as well as the one obtained in equation 10 below enables us to extend the fast marching method algorithm to a much wider class than was done before. For example, using this observation we were able to solve an etching problem, also considered in [3] where the authors did not use a fast marching method algorithm, but instead used a local narrow band approach and schemes devised in [64]. The Hamiltonian was

$$H(\varphi_x, \varphi_y, \varphi_z) = \sqrt{\varphi_z^2}\left(1 + \frac{4(\varphi_x^2 + \varphi_y^2)}{\varphi_x^2 + \varphi_y^2 + \varphi_z^2}\right).$$

We are able to use the same heap sort technology as for the eikonal equation, for problems of this type. See figures 3 and 4. These figures represent the level contours of an etching process whose normal velocity is a function of the direction of the normal. The process moves down in figure 3 and up in figure 4.

More generally, for $H(u,v)$ having the property

$$uH_1 \geq 0, \quad vH_2 \geq 0 \quad (9)$$

then we also proved [61]

$$H^G(u_+, u_-; v_+, v_-) = \max[H(u_+^-, v_+^-), H(u_-^+, v_+^-), H(u_+^-, v_-^+), H(u_-^+, v_-^+)] \quad (10)$$

and property $\mathcal{F}$ is again satisfied.

Again in [61], we were able to solve a somewhat interesting and very anisotropic etching problem with this new fast algorithm. Here we took

$$H(\varphi_x, \varphi_y) = |\varphi_y|(1 - a(\varphi_y)\varphi_y/(\varphi_x^2 + \varphi_y^2))$$

where

$$\begin{aligned} a &= 0 \quad \text{if} \quad \varphi_y < 0 \\ a &= .8 \quad \text{if} \quad \varphi_y > 0 \end{aligned}$$

and observed merging of two fronts. See figures 5 and 6. These figures show a two dimensional etching process resulting in a merger.

The fast method originating in [86] is a variant of Dijkstra's algorithm and, as such involves the tree like heap sort algorithm in order to compute the smallest of a set of numbers. Recently Boué and Dupuis [11] have proposed an extremely simple fast algorithm for a class of convex Hamiltonians including those which satisfy (4) and (5) above. Basically, their statement is that the standard Gauss-Seidel algorithm, with a simple ordering, converges in a *finite* number of iterations for equation (7). This would give an $O(N)$, not $O(N \log N)$ operations, with an extremely simple to program algorithm – no heap sort is needed. Moreover, for the eikonal equation with $a(x, y) = 1$, the algorithm would seem to converge in $2^d N$ iterations in R^d, $d = 1, 2, 3$, which is quite fast. This gives a very simple and fast redistancing algorithm. For more complicated problems we have found more iterations to be necessary, but still obtained promising results, together with some theoretical justification. See [85] for details, which also include results for a number of nonconvex Hamiltonians. We call this technique the "fast sweeping method" in [85]. We refer to it in section 3 when we discuss the basic distance reinitialization algorithm.

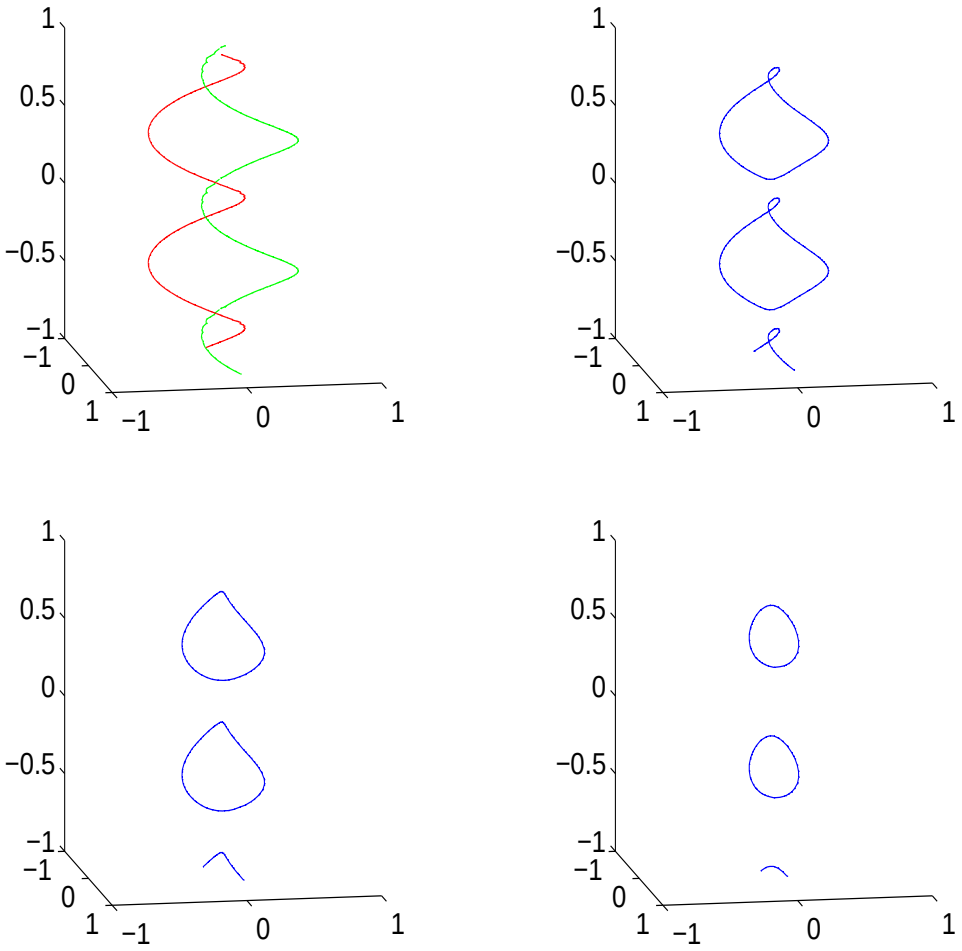


Figure 1: Merging and pinching of curves in R^3 moving by mean curvature. Reprinted from [13].

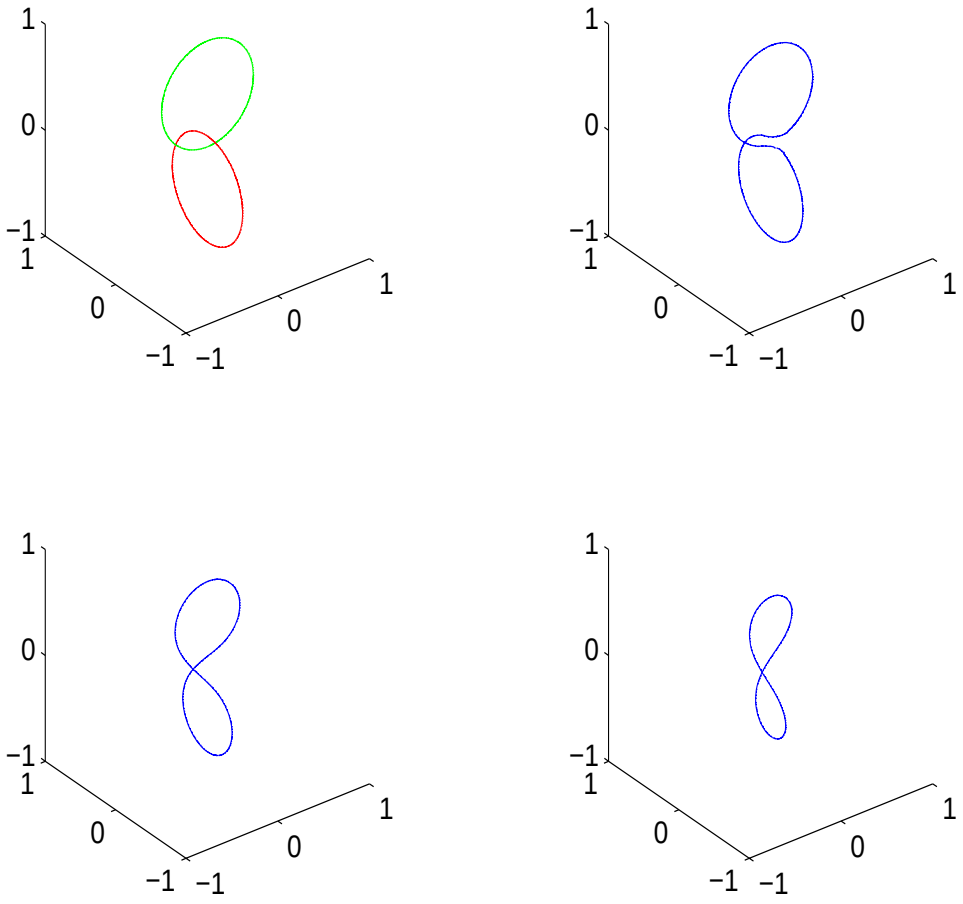


Figure 2: Merging and pinching of curves in R^3 moving by mean curvature. Reprinted from [13].

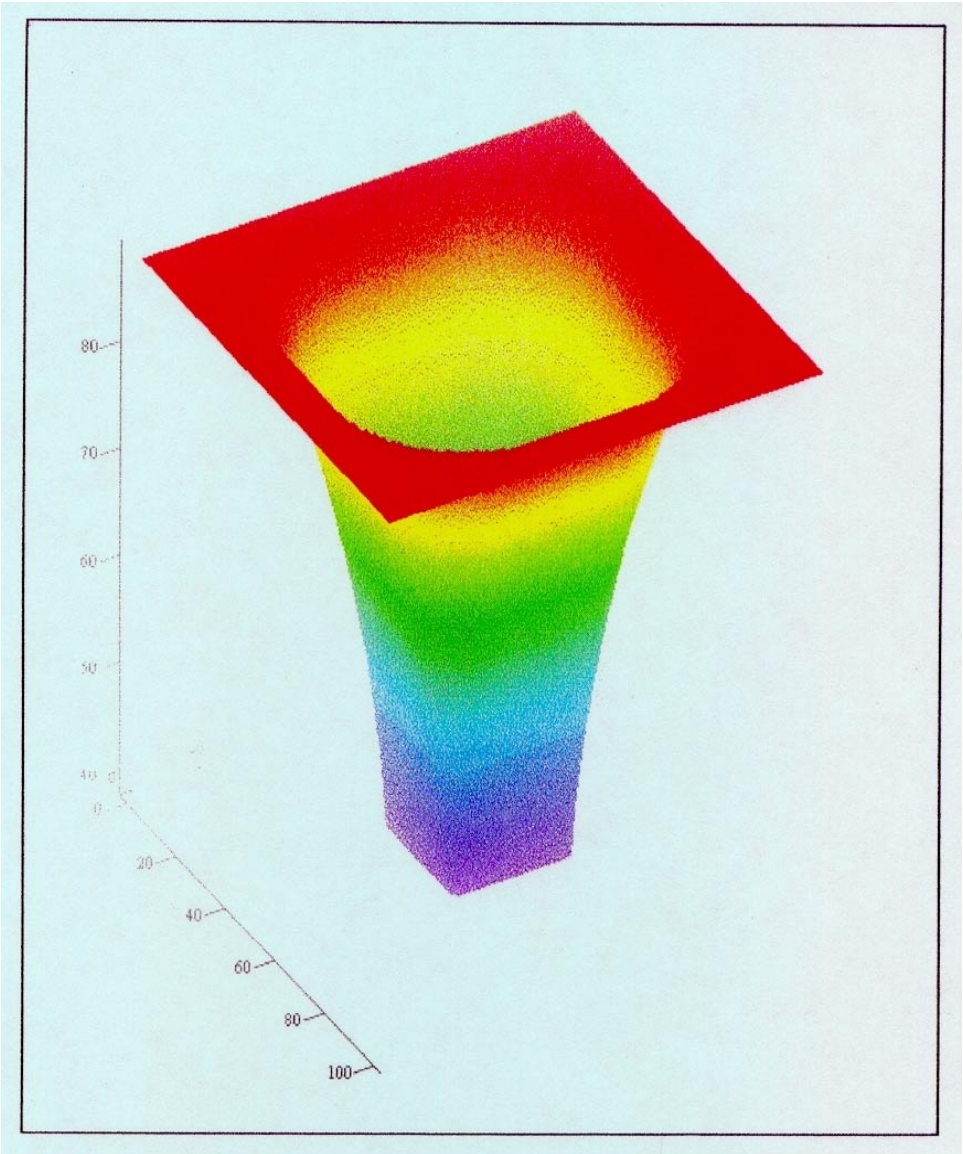


Figure 3: Three dimensional etching using a fast algorithm. Reprinted from [61].

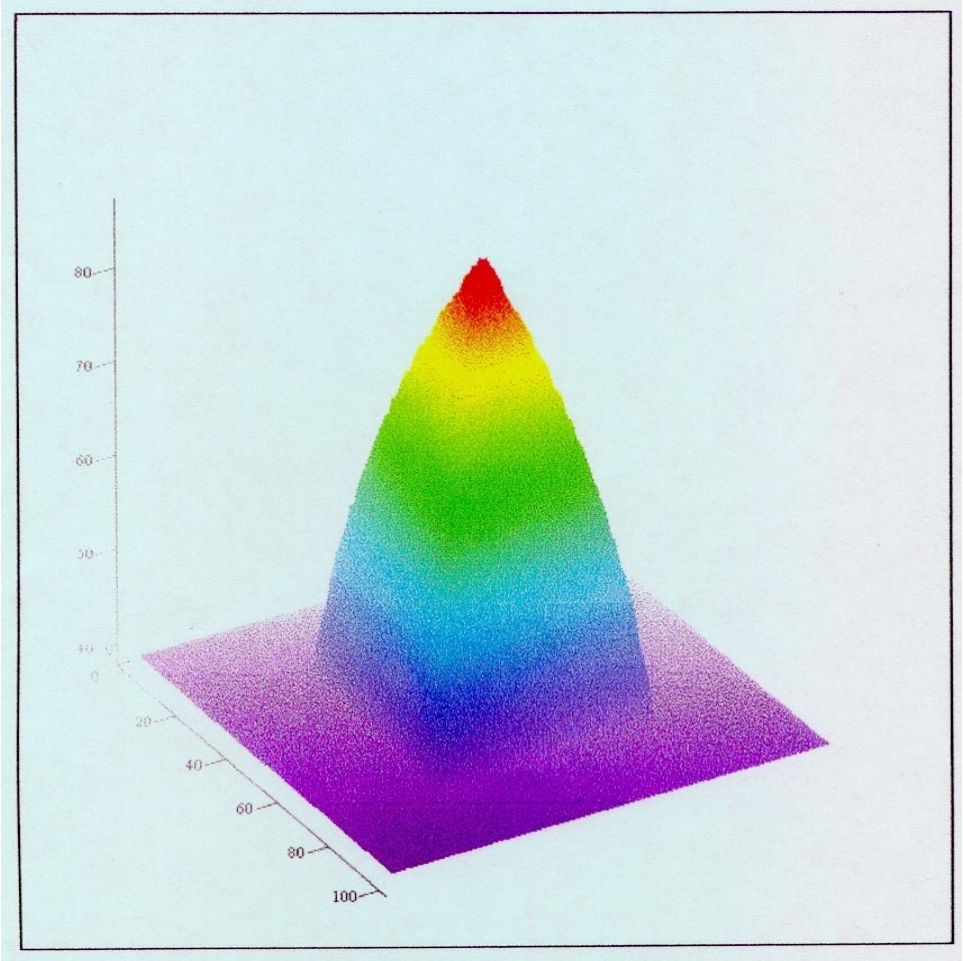


Figure 4: Three dimensional etching using a fast algorithm. Reprinted from [61].

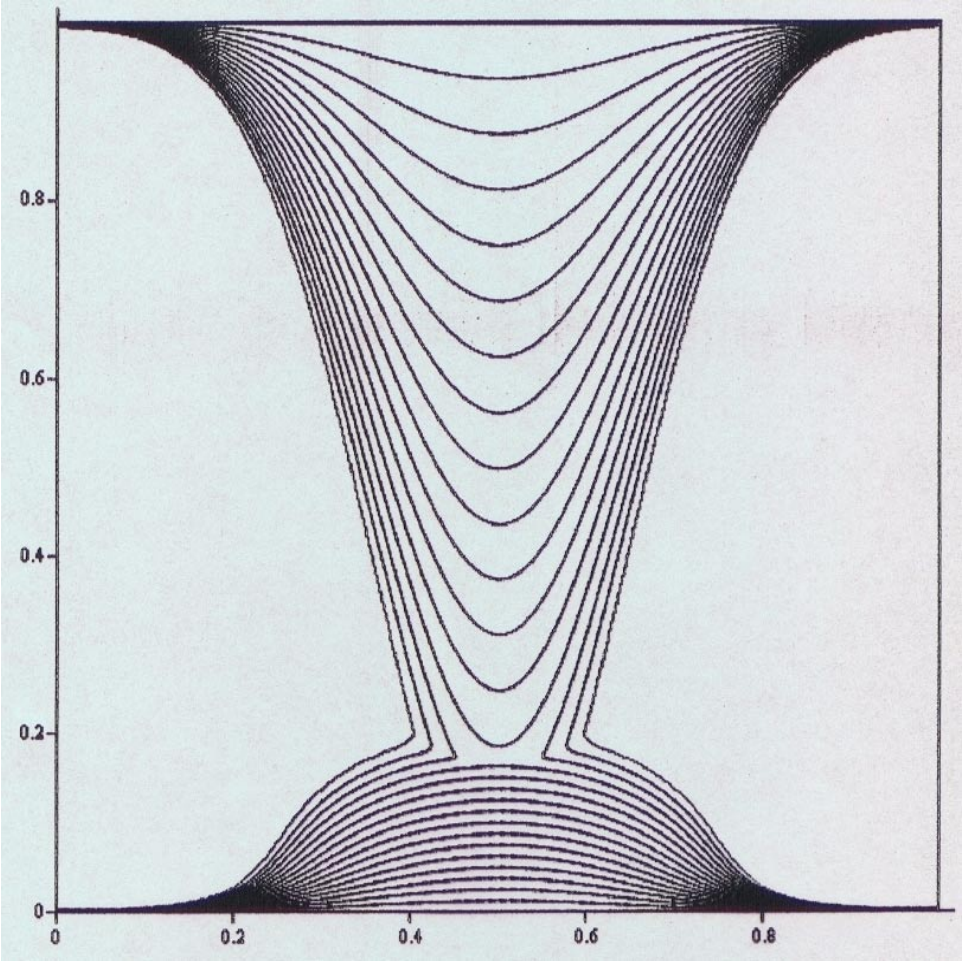


Figure 5: Two dimensional etching with merging using a fast algorithm. Reprinted from [61].

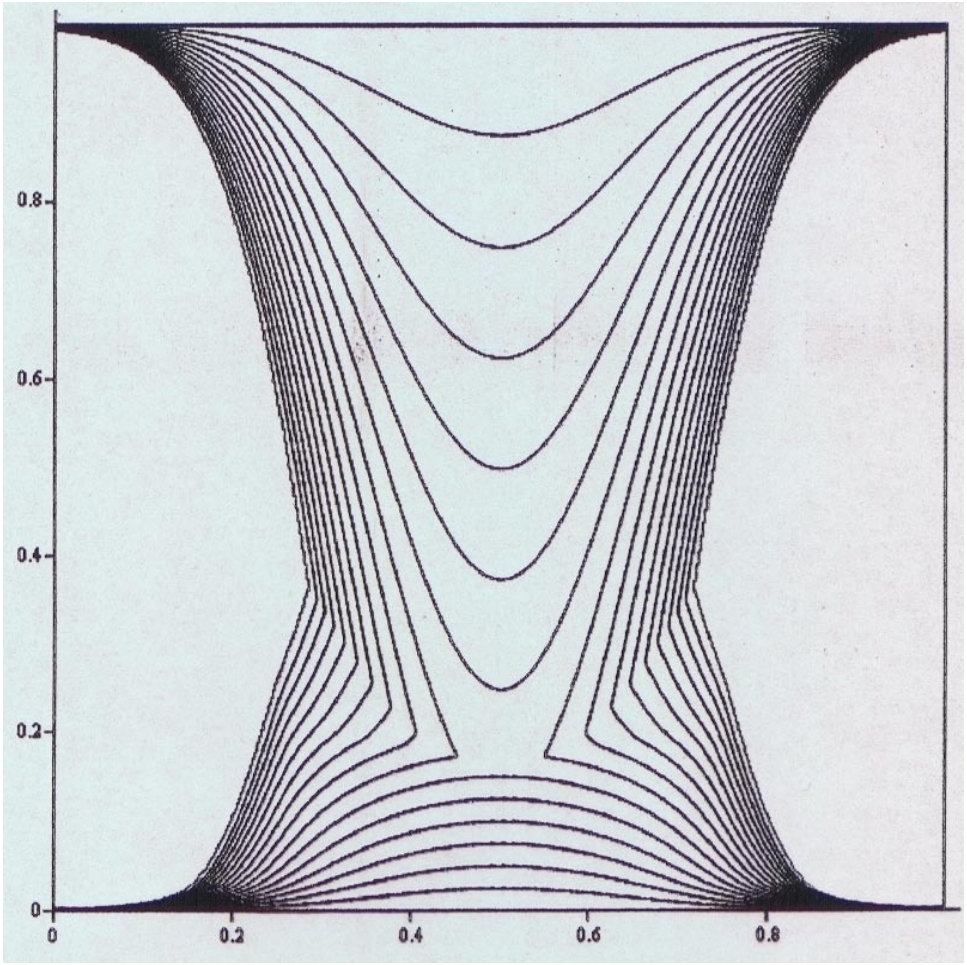


Figure 6: Two dimensional etching with merging using a fast algorithm. Reprinted from [61].

3 Level Set Dictionary, Technology and Numerical Implementation

We list key terms and define them by their level set representation.

1. The interface boundary $\Gamma(t)$ is defined by: $\{\vec{x}|\varphi(\vec{x},t)=0\}$. The region $\Omega(t)$ is bounded by $\Gamma(t)$: $\{\vec{x}|\varphi(\vec{x},t)>0\}$ and its exterior is defined by: $\{\vec{x}|\varphi(\vec{x},t)<0\}$

2. The unit normal $\vec{N}$ to $\Gamma(t)$ is given by

$$\vec{N}=-\frac{\nabla\varphi}{|\nabla\varphi|}.$$

3. The mean curvature κ of $\Gamma(t)$ is defined by

$$\kappa=-\nabla\cdot\left(\frac{\nabla\varphi}{|\nabla\varphi|}\right).$$

4. The Dirac delta function concentrated on an interface is:

$$\delta(\varphi)|\nabla\varphi|,$$

where $\delta(x)$ is a one dimensional delta function.

5. The characteristic function χ of a region $\Omega(t)$:

$$\chi=H(\varphi)$$

where

$$\begin{aligned} H(x) &\equiv 1 \text{ if } x>0 \\ H(x) &\equiv 0 \text{ if } x<0. \end{aligned}$$

is a one dimensional Heaviside function.

6. The surface (or line) integral of a quantity $p(\vec{x},t)$ over Γ:

$$\int_{R^n} p(\vec{x},t)\delta(\varphi)|\nabla\varphi|d\vec{x}.$$

7. The volume (or area) integral of $p(\vec{x},t)$ over Ω

$$\int_{R^n} p(\vec{x},t)H(\varphi)d\vec{x}.$$

Next we describe three key technological advances which are important in many, if not most, level set calculations.

8. The distance reinitialization procedure replaces a general level set function $\varphi(\vec{x},t)$ by $d(\vec{x},t)$ which is the value of the distance from $\vec{x}$ to $\Gamma(t)$, positive outside and negative inside. This assures us that φ does not become too flat or too steep near $\Gamma(t)$. Let $d(\vec{x},t)$, be signed distance of $\vec{x}$ to the closest point on Γ. The quantity $d(\vec{x},t)$ satisfies $|\nabla d| = 1,\ d > 0$ in $\Omega, d < 0$ in $(\bar{\Omega})^c$ and is the steady state solution (as $\tau \to \infty$) to

$$\begin{aligned} \frac{\partial \psi}{\partial \tau} + \mathrm{sgn}(\varphi)(|\nabla \psi| - 1) &= 0 \\ \psi(\vec{x},0) &= \varphi(\vec{x},t). \end{aligned} \tag{11}$$

where $\mathrm{sgn}(x) = 2H(x)-1$ is the one dimensional signum function. This was designed in [84]. The key observation is that in order to define d in a band of width ϵ around Γ, we need solve (11) only for $\tau = O(\epsilon)$. It can easily be shown that this can be used globally to construct distance (with arbitrary accuracy) in $O(N \log N)$ iterations [66]. Alternatively, we may use Tsitsiklis' fast algorithm [86], which is also $O(N \log N)$, with a much smaller constant, but which is only first order accurate. A locally second order accurate (in the high resolution sense) fast marching method was proposed in [77]. While this method has a much lower local truncation error than a purely first order accurate method, it is still globally first order accurate except for special cases. Finally, we might also use the fast sweeping method from [11] and [85] as described in the last section, which appears to have $O(N)$ complexity and which is also only first order accurate, although this complexity estimate has not been rigorously justified.

9. Smooth extension of a quantity, e.g. v_n on Γ to a neighborhood of Γ. Let the quantity be $p(\vec{x},t)$. Solve to steady state ($\tau \to \infty$)

$$\begin{aligned} \frac{\partial q}{\partial \tau} &+ \mathrm{sgn}(\varphi)\left(\frac{\nabla \varphi}{|\nabla \varphi|} \cdot \nabla q\right) = 0 \\ q(\vec{x},0) &= p(\vec{x},t). \end{aligned}$$

Again, we need only solve this for $\tau = O(\epsilon)$ in order to extend p to be constant in the direction normal to the interface in a tube of width ϵ.

This was first suggested and implemented in [24], analyzed carefully in [88], and further discussed and implemented in both [32] and [66]. A computationally efficient algorithm based on heap sort technology and fast marching methods was devised in [1]. There are many reasons to extend a quantity off of Γ, one of which is to obtain a well conditioned normal velocity for level contours of φ close to $\varphi = 0$ [24]. Others involve implementation of the Ghost Fluid Method of [32] discussed in the next section.

10. The basic level set method concerns a function $\varphi(\vec{x}, t)$ which is defined throughout space. Clearly this is wasteful if one only cares about information near the zero level set. The local level set method defines φ only near the zero level set. We may solve (2) in a neighborhood of Γ of width $m\Delta x$, where m is typically 5 or 6. Points outside of this neighborhood need not be updated by this motion. This algorithm works in "φ" space – so not too much intricate computer science is used. For details see [66]. Thus this local method works easily in the presence of topological changes and for multiphase flow. An earlier local level set approach called "narrow banding" was devised in [2].

Finally, we repeat that, in the important special case where v_N in equation 2 is a function only of $\vec{x}$, t and $\nabla\varphi$ (e.g. $v_N = 1$), then equation 2 becomes a Hamilton-Jacobi equation whose solutions generally develop kinks (jumps in derivatives). We seek the unique viscosity solution. Many good references exist for this important subject, see e.g. [8, 27]. The appearance of these singularities in the solution means that special, but not terribly complicated, numerical methods have to be used, usually on uniform Cartesian grids. This was first discussed in [64] and numerical schemes developed there were generalized in [65] and [43]. The key ideas involve monotonicity, upwind differencing, essentially nonoscillatory (ENO) schemes and weighted essentially nonoscillatory (WENO) schemes. See [64], [65] amd [43] for more details.

4 Coupling of the Level Set Method with External Physics

Interface problems involving external physics arise in various areas of science. The computation of such problems has a very long history. Methods of choice include front tracking, see e.g. [87] and [41], phase-field methods, see e.g. [48] and [59], and the volume of fluid (VOF) approach, see e.g. [60] and [12]. The level set method has had major successes in this area. Much of the level set technology discussed in the previous two sections was developed with such applications in mind.

Here, we shall describe level set approaches to problems in compressible flow, incompressible flow, flows having singular vorticity, Stefan problems, kinetic crystal growth and a relatively new island dynamics model for epitaxial growth of thin films. We shall also discuss a recently developed technique, the ghost fluid method (GFM), which can be used (1) to remove numerical smearing and nonphysical oscillations in flow variables near the interface and (2) to simplify the numerical linear algebra arising in some of the problems in this section and elsewhere.

4.1 Compressible Flow

Chronologically, the first attempt to use the level set method in this area came in two phase inviscid compressible flow, [55]. There, to the equations of conservation of mass, momentum and energy, we appended equation (1), which we rewrote in conservation form as

$$(\rho\varphi)_t + \nabla \cdot (\rho\varphi\vec{v}) = 0 \tag{12}$$

using the density of the fluid ρ.

The sign of φ is used to identify which gas occupied which region, so it determines the local equation of state. This (naive) method suffered from spurious pressure oscillations at the interface, as shown in [46] and [45]. These papers proposed a new method which reduced these errors by using a nonconservative formulation near the interface. However, [46] and [45] still smear out the density across the interface, leading to terminal oscillations for many equations of state.

A major breakthrough in this area came in the development of the ghost fluid method (GFM) in [32]. This enables us to couple the level set representation of discontinuities to finite difference calculations of compressible

flows. The approach was based on using the jump relations for discontinuities which are tracked using equation (1) (for two phase compressible flow). What the method amounts to (in any number of space dimensions) is to populate cells next to the interface with "ghost values", which, for two phase compressible flow retain their usual values of pressure and normal velocity (quantities which are continuous across the interface), with extrapolated values of entropy and tangential velocity (which jump across the interface). These quantities are used in the numerical flux when "crossing" an interface.

An important aspect of the method is its simplicity. There is no need to solve a Riemann problem normal to the interface, consider the Rankine-Hugoniot jump conditions, or solve an initial-boundary value problem. Another important aspect is its generality. The philosophy appears to be: at a phase boundary, use a finite difference scheme which takes only values which are continuous across the interface, using the natural values whenever possible. Of course, this implies that the tangential velocity is treated in the same fashion as the normal velocity and the pressure when viscosity is present. The same holds true for the temperature in the presence of thermal conductivity.

Figure 7 shows results obtained for two phase compressible flow using the GFM together with the level set method. Air with density around $1\frac{kg}{m^3}$ is to the left of the interface and water with density around $1000\frac{kg}{m^3}$ is to the right of the interface. Note that there is no numerical smearing of the density at the interface itself which is fortunate as water cavitates at a density above $999\frac{kg}{m^3}$ leading to host of nonphysical problems near the interface. Note too, that the pressure and velocity are continuous across the interface, although there are kinks in both of these quantities. A more complicated multidimensional calculation is shown in figure 8 where a shock wave in air impinges upon a helium droplet. See [32] for more details.

While the GFM was originally designed for multiphase compressible flow, it can be generalized to treat a large number of flow discontinuities. In [33], we generalized this method to treat shocks, detonations and deflagrations in a fashion that removes the numerical smearing of the discontinuity. Figure 9 shows the computed solution for a detonation wave. Note that there is no numerical smearing of the leading wave front which is extremely important when trying to eliminate spurious wave speeds for stiff source terms on coarse grids as first pointed out by [26]. While shocks and detonations have associated Riemann problems, the Riemann problem for a compressible flow deflagration discontinuity is not well posed unless the speed of the deflagration is given. Luckily, there is a large amount of literature on the

G-equation for flame discontinuities which was originally proposed in [50]. The G-equation represents the flame front as a discontinuity in the same fashion as the level set method so that one can easily consult the abundant literature on the G-equation to obtain deflagration speeds for the Ghost Fluid Method. Figure 10 shows two initially circular deflagration fronts that have just recently merged together. Note that the light colored region surrounding the deflagration fronts is a precursor shock wave that causes the initially circular deflagration waves to deform as they attempt to merge.

The GFM was extended in [34] in order to treat the two phase compressible viscous Navier Stokes equations in a manner that allows for a large jump in viscosity across the interface. This paper spawned the technology needed to extend the GFM to multiphase incompressible flow including the effects of viscosity, surface tension and gravity as discussed in the next subsection.

4.2 Incompressible Flow

The earliest real success in the coupling of the level set method to problems involving external physics came in computing two-phase Navier-Stokes incompressible flow [84], [22]. The equations can be written as:

$$\begin{aligned} \vec{u}_t + \vec{u} \cdot \nabla \vec{u} + \frac{\nabla p}{\rho} &= \vec{g} + \frac{\nabla \cdot (2\mu D)}{\rho} + \frac{\delta(\varphi)\sigma\kappa\vec{N}}{\rho} \\ \nabla \cdot \vec{u} &= 0 \end{aligned}$$

where $\vec{u} = (u, v, w)$ is the fluid velocity, p is the pressure, $\rho = \rho(\varphi)$ and $\mu = \mu(\varphi)$ are the piecewise constant fluid densities and viscosities, g is the gravitational force, D is the viscous stress tensor, σ is the surface tension coefficient, κ is the curvature of the interface, $\vec{N}$ is the unit normal and $\delta(\varphi)$ is a delta function. See [87] and [12] for earlier front tracking and VOF methods (respectively) using a similar formulation. This equation is coupled to the front motion through the level set evolution equation (1) with $\vec{v} = \vec{u}$. This involves defining the interface numerically as having a finite width of approximately 3 to 5 grid cells. Within this smeared out band, the density, viscosity and pressure are modeled as continuous functions. Then the $\frac{\sigma\kappa\vec{N}}{\rho}$ term is used to approximate the surface tension forces which are lost when using a continuous pressure [84]. Successful computations using this model were performed in [84] and elsewhere [22]. Problems involving area loss were observed and significant improvements were made in [83].

As mentioned above, the technology from [34] motivated the extension of the Ghost Fluid Method to this two phase incompressible flow problem.

First, a new boundary condition capturing approach was devised and applied to the variable coefficient Poisson equation to solve problems of the form

$$\nabla \left(\frac{1}{\rho} \nabla p \right) = f$$

where the jump conditions $[p] = g$ and $[\frac{1}{\rho} \nabla p \cdot \vec{N}] = h$ are given and ρ is discontinuous across the interface. This was accomplished in [49]. A sample calculation from [49] is shown in figure 11 where one can see that both the solution, p, and its first derivatives are sharp across the interface without numerical smearing. Next, this new technique was applied to multiphase incompressible flow in [44]. Here, since one can model the jumps in pressure directly, there is no need to add the $\frac{\sigma \kappa \vec{N}}{\rho}$ source term to the right hand side of the momentum equation in order to capture the surface tension forces. Instead surface tension is modeled directly by imposing a pressure jump across the interface. In addition, [44] allows for exact jumps in both ρ and μ so that the nonphysical finite width smeared out interface in [84] can be replaced by a sharp interface. A three dimensional calculation of an (invisible) solid sphere impacting water causing a splash is shown in figure 12. Here the air has density near $1 \frac{kg}{m^3}$ while the water has density near $1000 \frac{kg}{m^3}$.

Recently, in [57], this boundary condition capturing technology was extended to treat two phase incompressible flames where the normal velocity is discontinuous across the interface as well. Figure 13 shows an example calculation where two flames have just merged. Note that the velocity vectors in figure 13 clearly indicate that the velocity is kept discontinuous across the flame front. [39] considered two phase incompressible flames as well, proposing a method that keeps the interface sharp and removes numerical smearing. Unfortunately, the method proposed in [39] cannot treat topological changes in the flame front. Our method improves upon [39] allowing flame front discontinuities to merge, as in figure 13, or pinch off. Figure 14 shows two flame fronts shortly after merging in three spatial dimensions.

4.3 Topological Regularization

In [37] and [38], it is shown that the level set formulation provides a new and novel way to regularize certain ill-posed equations of interface motion, by blocking interface self-intersection. We computed two and three dimensional unstable vortex motion without regularization other than that in the

discrete approximation to $\delta(\varphi)$ – this is done over a few grid points. The key observation is that viewing a curve or surface as the level set of a function, and then evolving it with a perhaps unstable velocity field, prevents certain types of blow up from occuring. This is denoted "topological regularization". For example a tracked curve can develop a figure eight pattern, but a level set captured curve will pinch off and stabilize before this happens. For the set up (involving two functions), see [37], where we perform calculations involving the Cauchy-Riemann equations. The motions agree until pinch off, when the topological stabilization develops.

As an example, we considered the two dimensional incompressible Euler equations, which may be written as

$$\begin{aligned} \omega_t + \vec{u} \cdot \nabla \omega &= 0 \\ \nabla \times \vec{u} &= \omega \\ \nabla \cdot \vec{u} &= 0 \end{aligned}$$

We are interested in situations in which the vorticity is initially concentrated on a set characterized by the level set function φ as follows

$$\text{Vortex patch: } \omega = H(\varphi)$$

$$\text{Vortex sheet: } \omega = \delta(\varphi), \ \text{(strength of sheet is } \frac{1}{|\nabla \varphi|}\text{)}$$

$$\text{Vortex sheet dipole: } \omega = \frac{d}{d\varphi}\delta(\varphi) = \delta'(\varphi).$$

The key observation is that φ also satisfies a simple advection equation and $\vec{u}$ and ω can be easily recovered. For example, for the vortex sheet case we solve

$$\begin{aligned} \varphi_t + \vec{u} \cdot \nabla \varphi &= 0 \\ \vec{u} &= \begin{pmatrix} -\partial y \\ \partial x \end{pmatrix} \Delta^{-1} \delta(\varphi). \end{aligned}$$

Standard Laplace solvers may be used. See [38] for results involving two and three dimensional flows. In [66] we added reinitialization and extension to this procedure and obtained improved results in the two dimensional case.

4.4 Stefan Problem

Another classical field concerns Stefan problems [24], see also [78] for an earlier, but much more involved level set based approach. Here we wish to simulate melting ice or freezing water, or more complicated crystalline growth, as in the island dynamics model discussed below.

We begin with a simplified, nondimensionalized model (see [47] for an extension as mentioned below),

$$\begin{aligned} \frac{\partial T}{\partial t} &= \nabla^2 T, \quad \vec{x} \not\in \partial\Omega = \Gamma(t) \\ v_N &= [\nabla T \cdot N], \quad \vec{x}\ \varepsilon\ \Gamma(t) \end{aligned}$$

where $[\cdot]$ denotes the jump across the boundary, and

$$T = -\bar{\varepsilon}_c \kappa (1 - A\cos(k_A\theta + \theta_0)) + \bar{\varepsilon}_v v_n (1 - A\cos(k_A\theta + \theta_0))$$

on $\Gamma(t)$, and where κ is the curvature, $\theta = \cos^{-1}\frac{\varphi_x}{|\nabla\varphi|}$, and the constants $A, k_A, \theta_0, \bar{\varepsilon}_c$, and $\bar{\varepsilon}_v$ depend on the material being modeled.

We directly discretize the boundary conditions at Γ: To update T at grid nodes near the boundary, if the stencil for the heat equation would cross Γ (as indicated by nodal sign change in φ), we merely use dimension by dimension one sided interpolation and the given boundary T value at an imaginary node placed at $\varphi = 0$ (found by interpolation on φ) to compute T_{xx} and or T_{yy}, (never interpolating across the interface) rather than the usual three point central stencils. The level set function φ is updated and then reinitialized to be equal to the signed distance to Γ. Note that the level set update uses v_N that has been extended off the interface. See [24] for details.

We note that one can easily extend this to

$$\frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T)$$

where k is a different positive constant inside and outside of Ω and

$$v_N = \left[k\nabla T \cdot \vec{N}\right], \ \vec{x}\ \varepsilon\ \Gamma(t).$$

as was recently done in [47].

An important observation is that our finite differencing at the interface leads to a nonsymmetric matrix inversion when applying implicit discretization in time, although the method does have nice properties such as second

order accuracy and a maximum principle. This lack of symmetry is a bit problematic for a fast implementation, especially for very large values of k. Fortunately, an extension of GFM can be used to derive a different spatial discretization producing a symmetric matrix that can be inverted rather easily using fast methods. This was originally proposed by Fedkiw [31] and is described below.

It is sufficient to explain how the spatial derivatives are derived with respect to one variable, since there are no mixed partial derivative terms. Suppose the interface point, x_f, falls in between two grid points x_i and x_{i+1}. From ϕ, the distances between x_i, x_{i+1} and x_f can be estimated by

$$x_f - x_i \approx \frac{-\phi_i \ \Delta x}{(\phi_{i+1} - \phi_i)} = \theta_1 \Delta x \tag{13}$$

$$x_{i+1} - x_f \approx \frac{\phi_{i+1} \ \Delta x}{(\phi_{i+1} - \phi_i)} = \theta_2 \Delta x \tag{14}$$

To avoid numerical errors caused by division by 0, θ_1 or θ_2 are not used if either is less than Δx^2. If $\theta_1 < \Delta x^2$, then x_f is assumed equal to x_i. If $\theta_2 < \Delta x^2$, then x_f is assumed equal to x_{i+1}. Either assumption is effectively a second order perturbation of the interface location leading to second order accurate spatial discretization. The nonsymmetric second order accurate discretization for T_{xx} given in [24] is

$$(T_{xx})_i \approx \frac{\left(\frac{T_f - T_i}{\theta_1 \Delta x}\right) - \left(\frac{T_i - T_{i-1}}{\Delta x}\right)}{\frac{1}{2}(\theta_1 \Delta x + \Delta x)} \tag{15}$$

$$(T_{xx})_{i+1} \approx \frac{\left(\frac{T_{i+2} - T_{i+1}}{\Delta x}\right) - \left(\frac{T_{i+1} - T_f}{\theta_2 \Delta x}\right)}{\frac{1}{2}(\Delta x + \theta_2 \Delta x)} \tag{16}$$

where T_f denotes the value of T at x_f and is determined from the boundary condition. Instead of using the nonsymmetric equations (15) and (16), Fedkiw [31] proposed using

$$(T_{xx})_i \approx \frac{\left(\frac{T_f - T_i}{\theta_1 \Delta x}\right) - \left(\frac{T_i - T_{i-1}}{\Delta x}\right)}{\Delta x} \tag{17}$$

$$(T_{xx})_{i+1} \approx \frac{\left(\frac{T_{i+2} - T_{i+1}}{\Delta x}\right) - \left(\frac{T_{i+1} - T_f}{\theta_2 \Delta x}\right)}{\Delta x} \tag{18}$$

which leads to a symmetric linear system when using implicit time discretization. Equation (17) is derived using linear extrapolation of T from one side

of the interface to the other, obtaining

$$T_G = T_f + (1 - \theta_1) \left(\frac{T_f - T_i}{\theta_1} \right) \tag{19}$$

as a ghost cell value for T at x_{i+1}. The standard second order discretization of $\frac{\partial^2 T}{\partial x^2}$ at x_i using T_G at x_{i+1} is

$$(T_{xx})_i \approx \frac{\left(\frac{T_G - T_i}{\Delta x} \right) - \left(\frac{T_i - T_{i-1}}{\Delta x} \right)}{\Delta x} \tag{20}$$

and the substitution of equation (19) into equation (20) leads directly to (17). Equation (18) is derived similarly.

Formulas (17) and (18) have $O(1)$ errors using formal truncation error analysis. However, they are second order accurate on a problem where the interface has been perturbed by $O(\Delta x^2)$, making them second order accurate in the interface location. Assume that the standard second order accurate discretization is used to obtain the standard linear system of equations for T at every grid point except for those adjacent to the interface, that is except for x_i and x_{i+1}. Since the linear system of equations for the nodes to the left and including x_i is independent of the system for the nodes to the right including x_{i+1}, only the linear system to the left is discussed here. Equation (20) is used to write a linear equation for T_i introducing a new unknown T_G, and the system is closed with equation (19) for T_G. In practice, equations (19) and (20) are combined to obtain equation (17) and a symmetric linear system of equations. This linear system of equations results in well determined values (up to some prescribed tolerance near roundoff error levels) of T at each grid node as well as a well determined value of T_G (from equation (19)). For the sake of reference, designate $\vec{T}$ as the solution vector containing the values of T at each grid point to the left and including x_i as well as the value of T_G at x_{i+1} which are obtained by solving this symmetric linear system. Below, $\vec{T}$ is shown to be a second order accurate solution to our problem by showing that it is the second order accurate solution to a modified problem where the interface location has been perturbed by $O(\Delta x^2)$.

Consider the modified problem where a Dirichlet boundary condition of $T = T_G$ is specified at x_{i+1} where T_G is chosen to be the value of T_G from $\vec{T}$ defined above. This modified problem can be exactly discretized to second order accuracy everywhere using the standard discretization at every node except x_i where equation (20) is used. We note that equation (20) *is* the

standard second order accurate discretization when a Dirichlet boundary condition of $T = T_G$ is applied at x_{i+1}. This new linear system can be discretized and solved in a standard fashion to obtain a second order accurate solution at each grid node. Then the realization that $\vec{T}$ is an exact solution to *this* linear system implies that $\vec{T}$ is a second order accurate solution to this modified problem. Next consider the interface location dictated by the modified problem. Since $\vec{T}$ is a second order accurate solution to the modified problem, $\vec{T}$ can be used to obtain the interface location to second order accuracy. The linear interpolant that uses T_i at x_i and T_G at x_{i+1} predicts an interface location of *exactly* x_f which is the true interface location. Since higher order interpolants (higher than linear) can contribute at most an $O(\Delta x^2)$ perturbation of the interface location, the interface location dictated by the modified problem is at most an $O(\Delta x^2)$ perturbation of the true interface location, x_f.

In [25], we used this strategy to obtain a second order accurate symmetric discretization of the variable coefficient Poisson equation

$$\nabla \left(k \nabla T \right) = f$$

on irregular domains in as many as three spatial dimensions. Then, in a straightforward way, we obtained second order accurate symmetric discretizations of the heat equation on irregular domains using backward Euler time stepping with $\triangle t = (\triangle x)^2$ and Crank-Nicolson time stepping with $\triangle t = \triangle x$.

4.5 Kinetic Crystal Growth

For an initial state consisting of any number of growing crystals in R^d, d arbitrary, moving outward with given normal growth velocity $\vec{v}(\vec{N}) > 0$ which depends on the angle of the unit surface normal $\vec{N}$, the asymptotic growth shape is a single (kinetic) Wulff-construct crystal. This result was first conjectured by Gross in (1918) [35]. This shape is also known to minimize the surface integral of $\vec{v}(\vec{N})$ for a given volume. We gave a proof of this result [62], see also [81], using the level set formulation and the Hopf-Bellman formulas [6] for the solution of a Hamilton-Jacobi equation. Additionally, with the help of the Brunn-Minkowski inequality, we showed that if we evolve a convex surface under the motion described in (3), then the ratio to be minimized monotonically decreases to its minimum as time

increases, which provides a new proof that the Wulff construction solves the generalized isoperimetric problem as well. Thus there is a new link between this hyperbolic surface evolution and this (generally nonconvex) energy minimization. This also provides a convenient framework for numerically computing anisotropic kinetic crystal growth [67]. The theoretical and numerical results of this work are illustrated in the Uniform Density Island Dynamics models of [15] and [36]. That model describes crystals growing in two dimensions with an anisotropic velocity.

An interesting spinoff of this work came in [67] in which we proved that any two dimensional Wulff shape can be interpreted precisely as the solution of a Riemann problem for a scalar conservation law – contact discontinuities correspond to jumps in the angle of the normal to the shape, smoothly varying non flat faces correspond to rarefaction waves and planar facets correspond to constant states, which develop because of kinks in the conservation law's flux function. These kinks are also seen in the convexified Wulff energy.

4.6 Epitaxial Growth of Thin Films

A new continuum model for the epitaxial growth of thin films has been developed. Molecular Beam Epitaxy (MBE) is a method for growing extremely thin films of material. The essential aspects of this growth process are as follows: under vacuum conditions a flux of atoms is deposited on a substrate material, typically at a rate that grows one atomic monolayer every several seconds. When deposition flux atoms hit the surface, they bond weakly rather than bounce off. These surface "adatoms" are relatively free to hop from lattice site to site on a flat (atomic) planar surface. However, when they hop to a site at which there are neighbors at the same level, they form additional bonds which hold them in place. This bonding could occur at the "step edge" of a partially formed atomic monolayer, which contributes the growth of that monolayer. Or, it could occur when two adatoms collide with each other. If the critical cluster size is one, the colliding adatoms nucleate a new partial monolayer "island" that will grow by trapping other adatoms at its step edges.

By these means, the deposited atoms become incorporated into the growing thin film. Each atomic layer is formed by the nucleation of many isolated monolayer islands, which then grow in area, merge with nearby islands, and ultimately fill in to complete the layer. Because the deposition flux is continually raining down on the entire surface, including the tops of the islands,

a new monolayer can start growing before the previous layer is completely filled. Thus islands can form on top of islands in a "wedding cake" fashion, and the surface morphology during growth can become quite complicated.

The Island Dynamics model is a continuum model designed to capture the longer length scale features that are likely to be important for the analysis and control of monolayer thin film growth. It is also intended to model the physics relevant to studying basic issues of surface morphology, such as the effects of noise on growth, the long time evolution of islands, and the scaling relationships between surface features (mean island area, step edge length, etc) in various growth regimes (precoalescence, coalescence). Refer to the classic work of [14] for useful background on the modeling of the growth of material surfaces. Our present discussion of the Island Dynamics Model is an abridged version of what was discussed in [54]. We shall present this new model in some detail because, although it has many of the features of the Stefan problem, it also requires some new level set technology. This includes a "wedding cake" formulation involving several level sets of the same function, nucleation of new islands, and nontrivial numerical treatment of the interface to obtain rapid convergence of implicit time marching schemes.

In the Island Dynamics model, we treat each of the islands present as having a unit height, but a continuous (step edge) boundary on the surface. This represents the idea that the films are atomic monolayers, so that height is discrete, but they cover relatively large regions on the substrate, so x and y are continuum dimensions. The adatoms are modeled by a continuous adatom density function on the surface. This represents the idea that they are very mobile, and so they effectively occupy a given site for some fraction of the time, with statistical continuity, rather than discretely.

Thus, the domain for the model is the $x-y$ region originally defined by the substrate, and the fundamental dynamical variables for this model are:

- The island boundary curves $\Gamma_i(t),\ i = 1, 2, \ldots, N$
- The adatom density on the surface $\rho(x, y, t)$

The adatom density ρ obeys a surface diffusive transport equation, with a source term for the deposition flux

$$\frac{\partial \rho}{\partial t} = \nabla \cdot (D \nabla \rho) + F,$$

where $F = F(x, y, t)$ is specified. During most phases of the growth, it is simply a constant. This equation may also include additional small loss

terms reflecting adatoms lost to the nucleation of new islands, or lost to de-absorption off the surface. This equation must be supplemented with boundary conditions at the island boundaries. In the simplest model of Irreversible Aggregation, the binding of adatoms to step edges leaves the adatom population totally depleted near island boundaries, and the boundary condition is

$$\rho|_\Gamma = 0.$$

More generally, the effects of adatom detachment from boundaries, as well as the energy barriers present at the boundary, lead to boundary conditions of the form

$$\left[A\rho + B\frac{\partial \rho}{\partial n}\right] = C$$

where C is given and $[\cdot]$ denotes the local jump across the boundary. In particular, note that ρ itself can have a jump across the boundary, even though it satisfies a diffusive transport equation. This simply reflects that fact that the adatoms on top of the island are much more likely to incorporate into the step edge than to hop across it and mix with the adatoms on the lower terrace, and vice versa.

The island boundaries Γ_i move with velocities $\vec{v} = v_N\vec{N}$, where the normal velocity v_n reflects the island growth. This is determined simply by local conservation of atoms: the total flux of atoms to the boundary from both sides times the effective area per atom, a^2, must equal the local rate of growth of the boundary, v_N:

$$v_N = -a^2[\vec{q}\cdot\vec{N}]$$

(this assumes there is no particle transport along the boundary; more generally, there is a contribution from this as well) where $\vec{q}$ is the surface flux of adatoms to the island boundary and $\vec{N}$ is the local outward normal. In general, the net atom flux $\vec{q}$ can be expressed in terms of the diffusive transport, as well as attachment and detachment probabilities, all of which can be directly related to the parameters of Kinetic Monte Carlo models. In the special case of Irreversible Aggregation, $\vec{q}$ is simply the surface diffusive flux of adatoms

$$\vec{q} = -D\nabla\rho.$$

To complete the model we include a mechanism for the nucleation of new islands. If islands nucleate by random binary collisions between adatoms

(and if the critical cluster size is one), we expect the probability that an island is nucleated at a time t, at a site (x, y), scales like

$$P[dx, dy, dt] = \epsilon\rho(x, y, t)^2 dt\ dx\ dy.$$

This model describes nucleation as a site-by-site, timestep-by-timestep random process. A simplifying alternative is to assume the nucleation occurs at the continuous rate obtained by averaging together the probabilistic rates at each site. In this case, if we let $n(t)$ denote the total number of islands nucleated prior to time t, we have the deterministic equation

$$\frac{dn}{dt} = \langle \epsilon\rho^2 \rangle$$

where $\langle \cdot \rangle$ denotes the spatial average. In this formulation, at each time when $n(t)$ reaches a new integer value, we nucleate a new island in space. This is carried out by placing it randomly on the surface with a probability weighted by ρ^2, so that the effect of random binary collisions is retained.

This basic model also has natural extensions to handle more complex thin film models. For example, additional continuum equations can be added to model the dynamics of the density of kink sites on the island boundaries, which is a microstructural property that significantly influences the local adatom attachment rates (see [15]). Also, we can couple this model to equations for the elastic stress that results from the "lattice mismatch" between the size of the atoms in the growing layers and the size of the atoms in the substrate.

Conversely, the above model has a particular interesting extreme simplification. We can go to the limit where the adatoms are so mobile on the surface ($D \to \infty$) that the adatom density is spatially uniform, $\rho(x, y, t) = \rho(t)$. In this case, the loss of adatoms due to the absorbing boundaries is assumed to take on a limiting form proportional to the adatom density and the total length L of all the island boundaries, which can be written as a simple sink term

$$\frac{d\rho}{dt} = F - \lambda L\rho.$$

(This equation can be derived systematically from the conservation law for the total number of adatoms, $\int \rho$, that follows from the adatom diffusion equation. The above loss term is just a simplified model for the net loss of adatoms to the island boundaries.) Further, it is assumed the velocity takes on a given normal dependent limiting form, $v_N = v_N(\vec{N})$ (which implies

that growing islands will rapidly assume the associated "Wulff shape" for this function $v_N(\vec{N})$ (as in [62])). We have used this "Uniform Density" model to prototype the numerical methods, and to develop an understanding of how the island dynamics models are related to the continuum "rate equation" models that describe island size distribution evolution while using no information at all about the spatial interactions of the islands.

Much of the above model is formally a Stefan problem and many of the level set techniques required for this were developed in [24] and can similarly be applied here. In addition, the internal boundary condition discretization of the adatom diffusion equation can be implemented using the symmetric matrix version of the discretization proposed by Fedkiw [31] and discussed earlier in this paper.

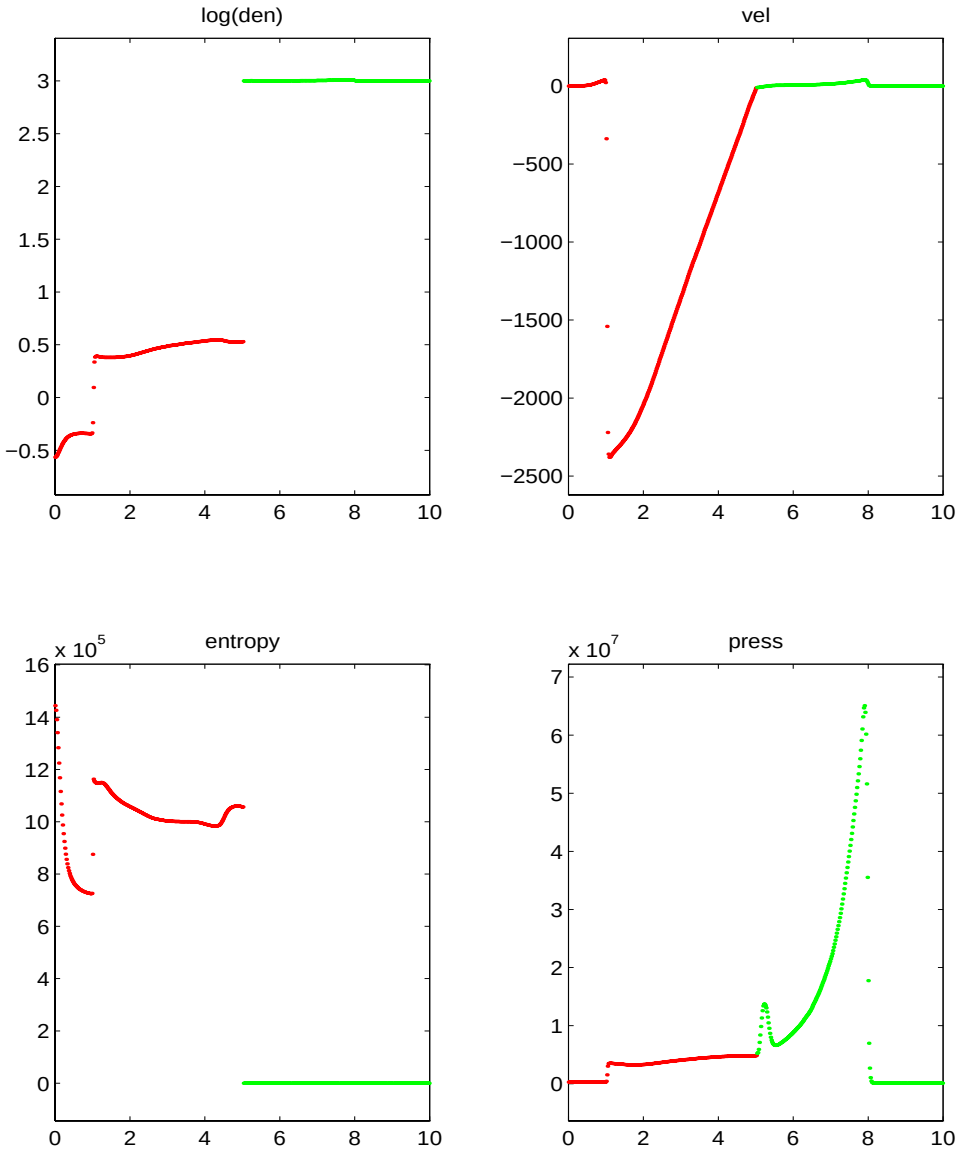


Figure 7: Two phase compressible flow calculated with the Ghost Fluid Method. Air on the left and water on the right. Reprinted from [32].

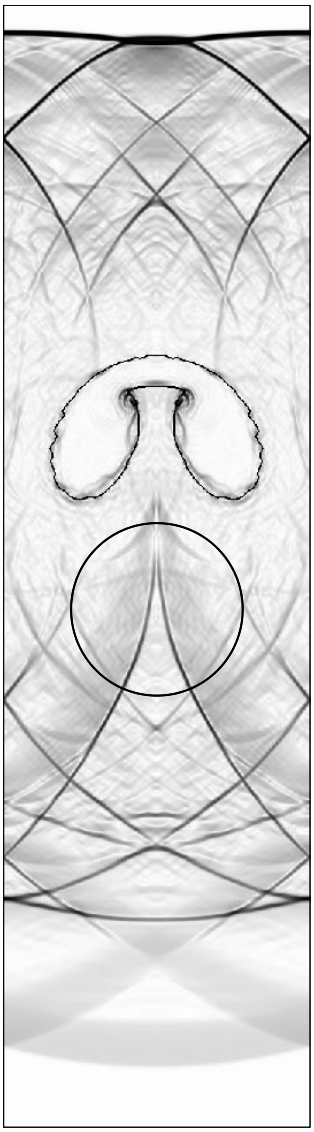


Figure 8: Mach 1.22 air shock collapse of a helium bubble. Reprinted from [32].

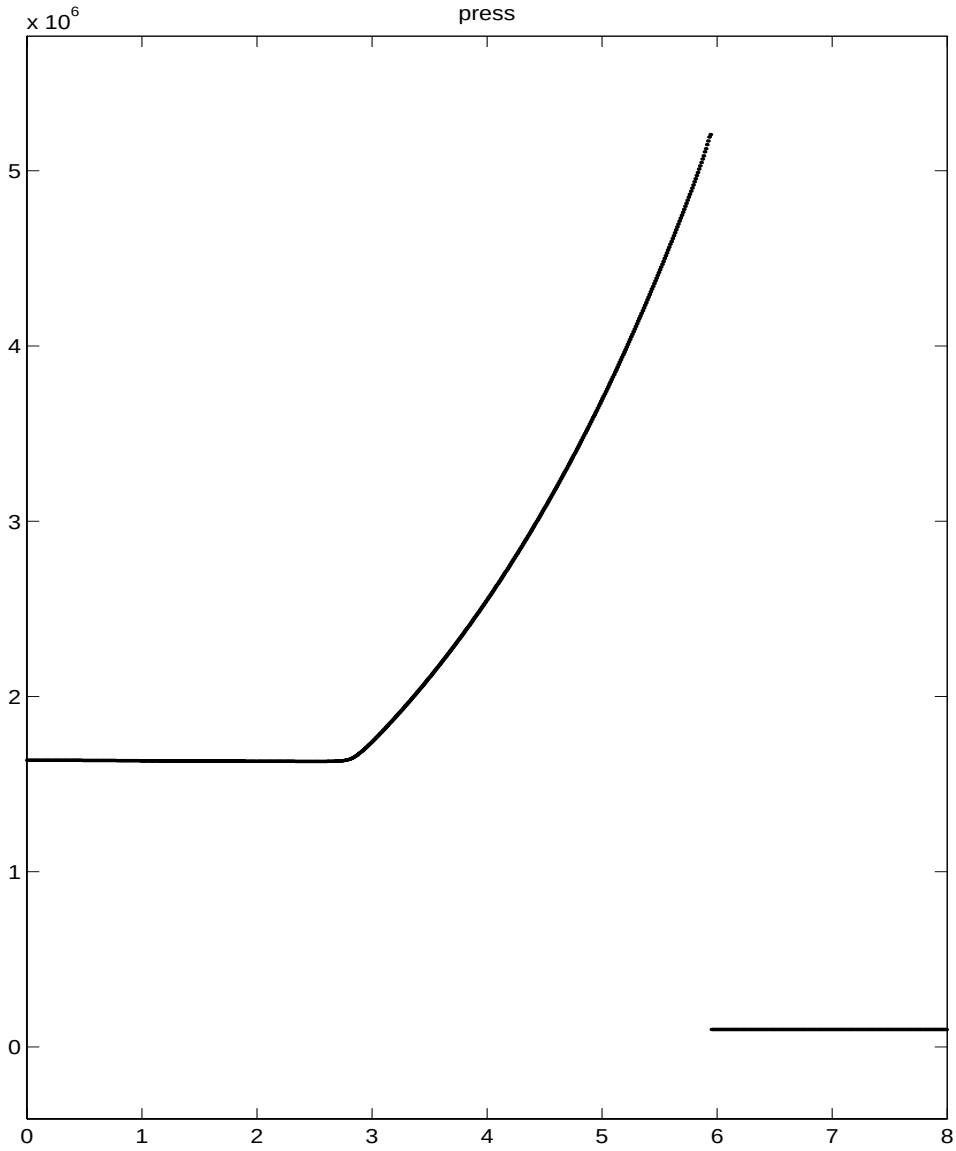


Figure 9: Nonsmeared detonation wave traveling away from a solid wall. Reprinted from [33].

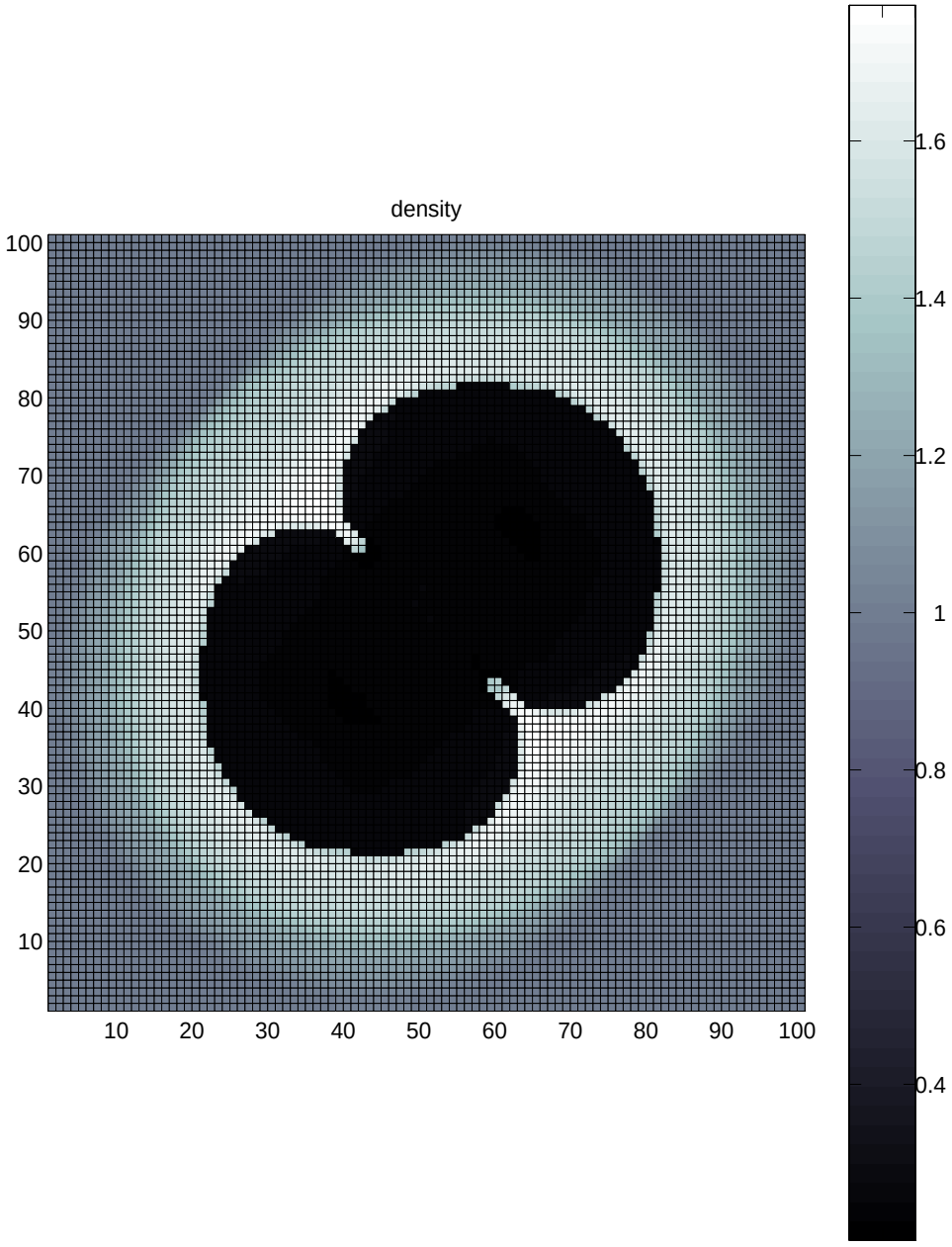


Figure 10: Two deflagration fronts depicted shortly after merging. Reprinted from [33].

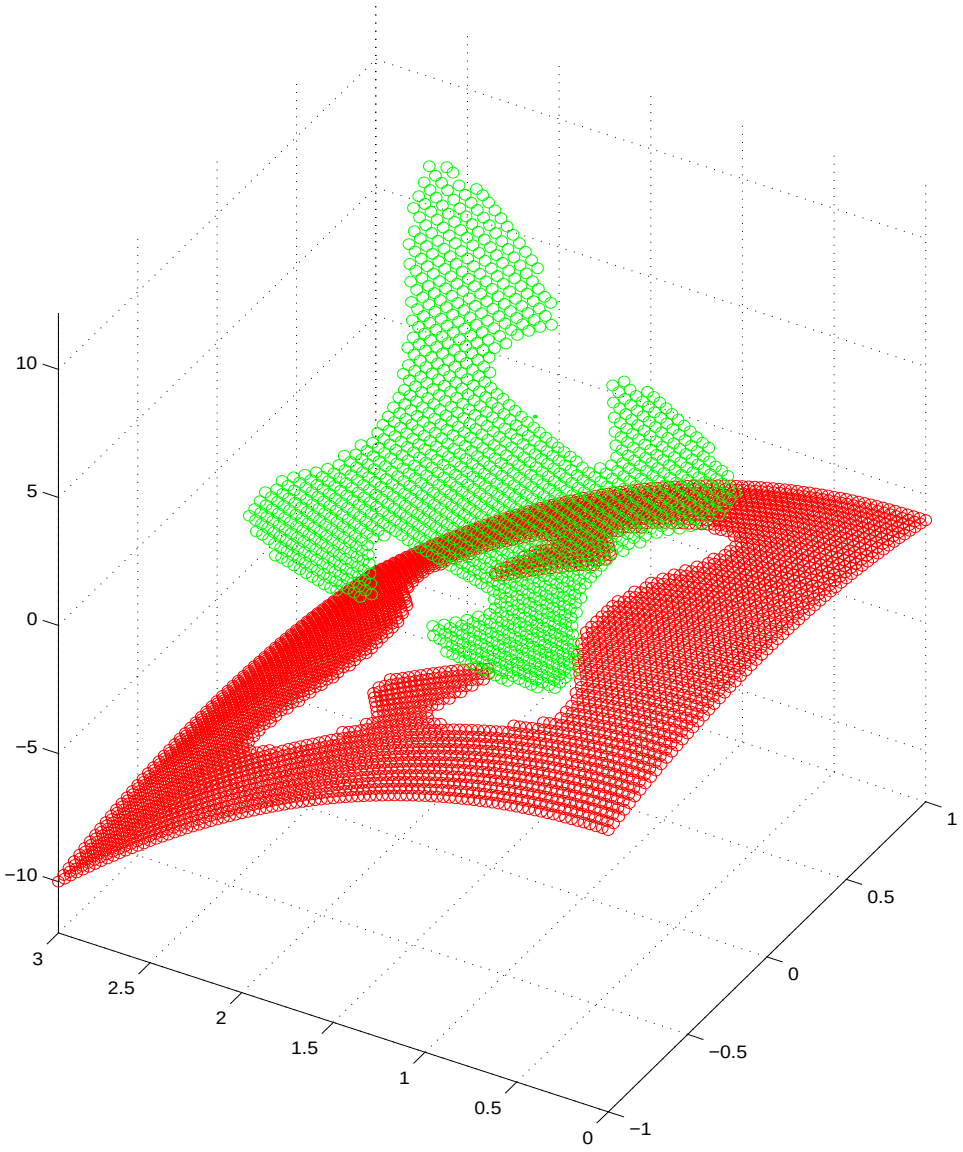


Figure 11: Two spatial dimensions, $\nabla \cdot (\frac{1}{\rho}\nabla p) = f(x,y)$, $[p] = g(x,y)$, $[\frac{1}{\rho}\nabla p \cdot \vec{N}] = h(x,y)$. Reprinted from [49].

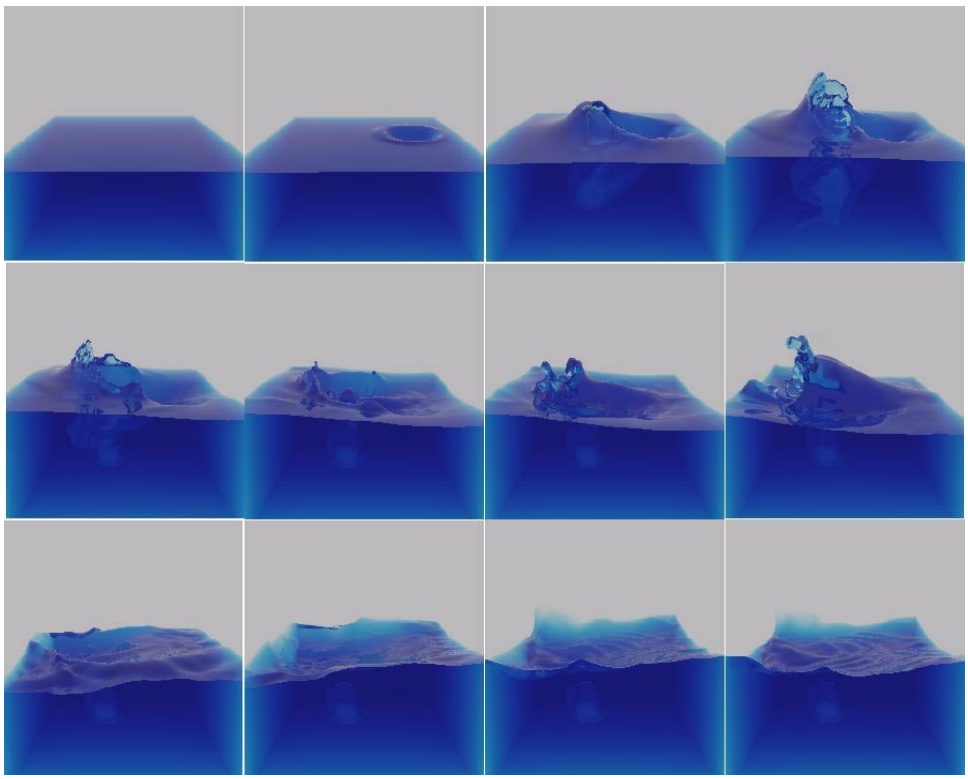


Figure 12: Water waves generated by the impact of an (invisible) solid object. Reprinted from [44].

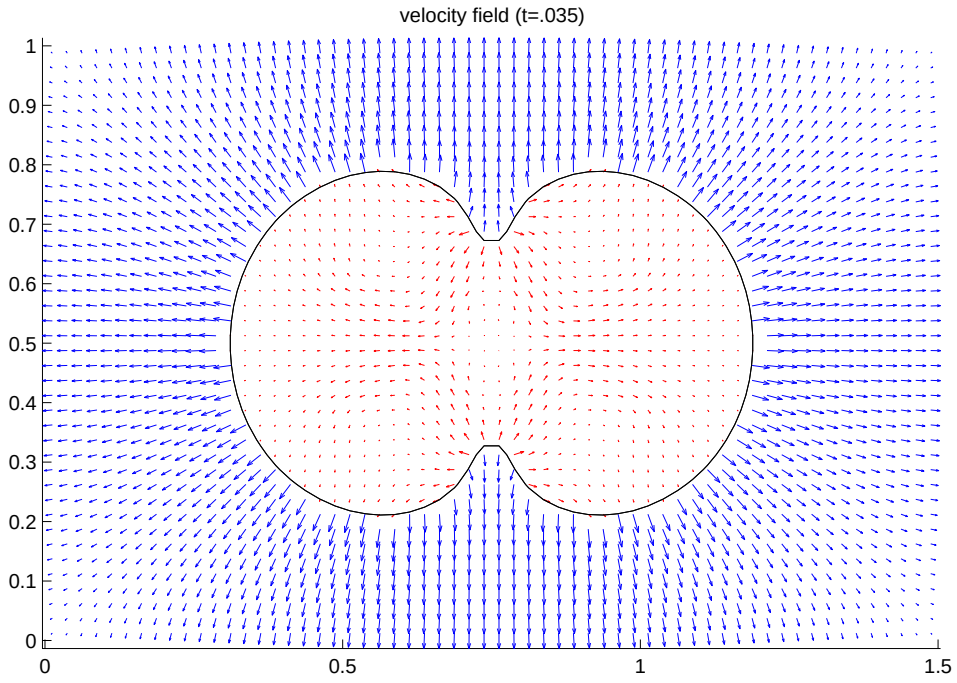


Figure 13: Two phase incompressible flames depicted shortly after merging (2 spatial dimensions). Reprinted from [57].

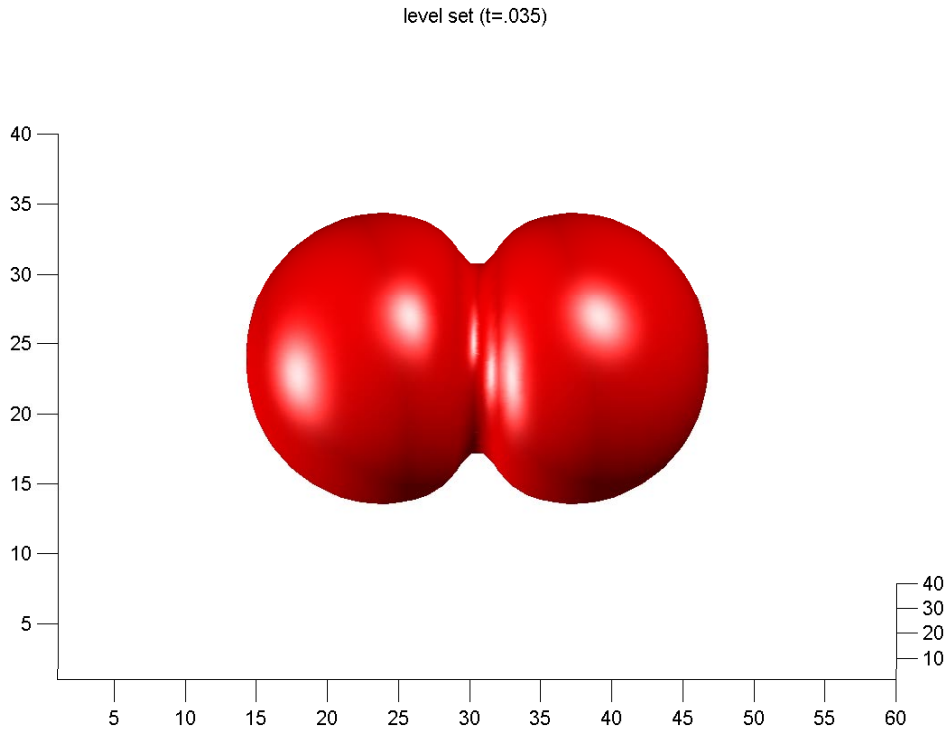


Figure 14: Two phase incompressible flames depicted shortly after merging (3 spatial dimensions). Reprinted from [57].

5 A Variational Approach with Applications to Multiphase Motion

In many situations, e.g., crystal growth, a material is composed of three or more phases. The interfaces between the phases move according to some law. If the material is a metal and its grain orientation is different in each region, then an isotropic surface energy means that the velocity is the mean curvature of the interface. Or the velocities of the interfaces may depend on the pair of phases in contact; e.g. a different constant velocity on each interface.

Several fixed grid approaches to this problem have been used. Merriman, Bence and Osher [53] have extended the level set method to compute the motion of multiple junctions. Also in that paper, and in [51] and [52], a simple method based on the diffusion of characteristic functions of each set Ω_i, followed by a certain reassignment step, was shown to be appropriate for the motion of multiple junctions in which the bulk energies are zero (and hence, the constants $e_i = 0,\ i = 1, \ldots, n$) and the $f_{i,j}$ are all equal to the same positive constant, i.e., pure mean curvature flow. See equations (21) below.

Another method using an "influence matrix" was designed in [75]. However, as cautioned by the author, the method is expensive and complex.

More general motion involving somewhat arbitrary functions of curvature, perhaps different for each interface, was proposed in [53] as well. This was implemented basically by decoupling the motions, and then using a reassignment step. Again each region has its own private level set function. This function moves each level set with a normal velocity depending on the proximity to the nearest interface, thus vacuum and overlapping regions generally develop. Then a simple reassignment step is used, removing all the spurious regions. For details see [53]. In that paper there was no restriction to gradient flows. However, the general method in [53] lacks (so far) a clean theoretical basis to guide the design of numerical algorithms. These difficulties were rectified by the following method.

In [88] we developed the variational level set approach inspired by [68]. Given a disjoint family Ω_i of regions in R^2 with the common boundary between Ω_i and Ω_j denoted by $\Gamma_{i,j}$, we associate to this geometry an energy function of the form

$$E \;=\; E_1 + E_2$$

$$
\begin{aligned}
E_1 &= \sum_{1\le i\le j\le n} f_{i,j} \text{ length } (\Gamma_{i,j}) \qquad (21)\\
E_2 &= \sum_{1\le i\le n} e_i \text{ area } (\Omega_i)
\end{aligned}
$$

where E_1 is the energy of the interface (surface tension). E_2 is bulk energy, and n is the number of phases. The gradient flow induces motion such that the normal velocity of each interface is defined in (22). At triple points (which can be seen geometrically by the triangle inequality to be the only stable junctions if all the $f_{i,j} > 0$), the angles are determined by (23) throughout the motion.

$$\text{Normal velocity of } \Gamma_{i,j} = (v_N)_{i,j} = f_{i,j}\kappa_{i,j} + (e_i - e_j). \qquad (22)$$

$$\frac{\sin\theta_1}{f_{2,3}} = \frac{\sin\theta_2}{f_{3,1}} = \frac{\sin\theta_3}{f_{1,2}}. \qquad (23)$$

This could be rewritten as:

$$
\begin{aligned}
E &= E_1 + E_2\\
E_1 &= \sum_{i=1}^{n} \gamma_i \int\int \delta(\varphi_i(x,y,t))|\nabla\varphi_i(x,y,t)|dxdy \qquad (24)\\
E_2 &= \sum_{i=1}^{n} e_i \int\int H(\varphi_i(x,y,t))dxdy,
\end{aligned}
$$

where

$$f_{i,j} = \gamma_i + \gamma_j, \quad 1 \le i < j \le n.$$

In the (most interesting) case when $n = 3$ we can solve uniquely for the γ_i.

Now our problem becomes:

Minimize E subject to the constraint that

$$\sum_{i=1}^{n} H(\varphi_i(x,y)) - 1 \equiv 0. \qquad (25)$$

This infinite set of constraints prevents the development of overlapping regions and/or vacuum. It requires that the level curves $\{(x,y)|\varphi_i(x,y,t) = 0\}$ match perfectly.

The implementation of (24) with the infinite set of constraints (25) is computationally demanding. Instead we try to replace the constraint (25) by a single constraint

$$\int\int \frac{(\sum H(\varphi_i(x,y,t)) - 1)^2}{2} dxdy = \epsilon \tag{26}$$

where $\epsilon > 0$ is as small as we can manage numerically.

The gradient projection method leads us to an interesting coupled system which involves motion of level contours of each φ with normal velocity $a + b\kappa$ together with a term enforcing the no overlap/vacuum constraint. We find that $\epsilon \approx \Delta x$ in real calculations. See [88] for details.

We have used this technique to reproduce the general behavior of complicated bubble and droplet motions in two and three dimensions [90]. The problems included soap bubble colliding and merging, drops falling or remaining attached to a generally irregular ceiling (see figure 15), liquid penetrating through an asymmetric funnel opening (see figure 16), and mercury sitting on the floor (see figure 17).

This variational approach has also been found to have many applications in computer vision – this will be discussed in the next section.

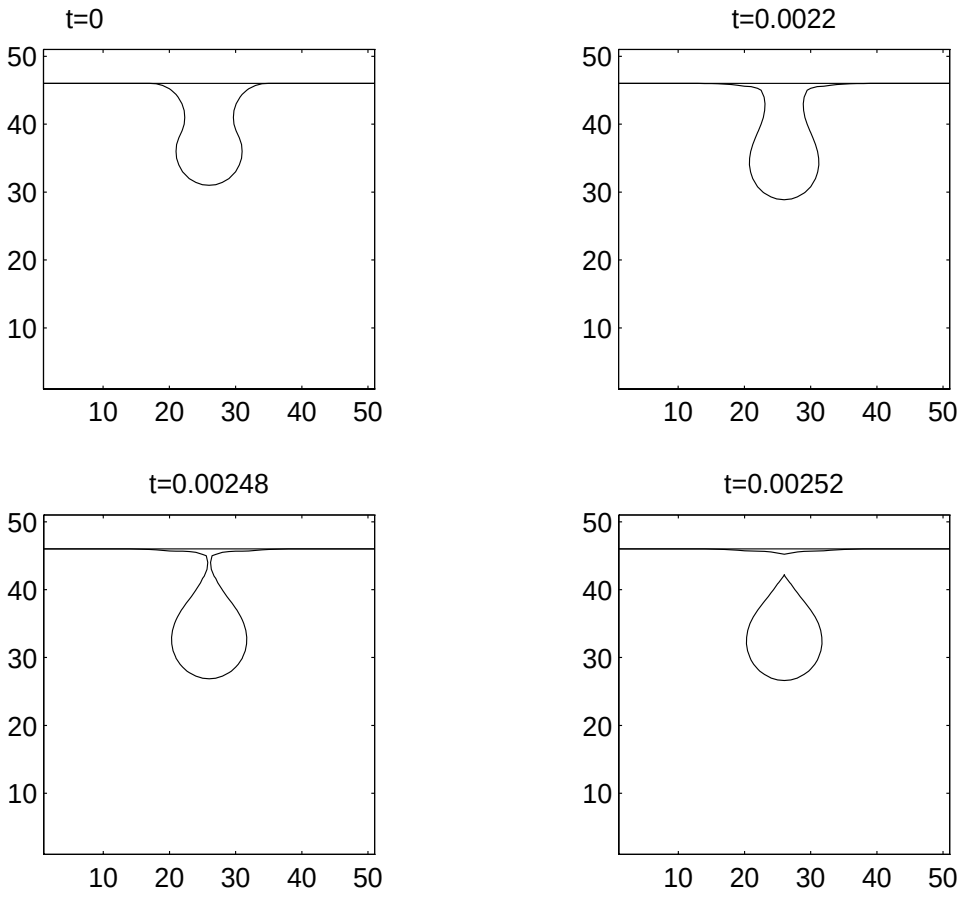


Figure 15: Three dimensional drop falling from ceiling. Reprinted from [90].

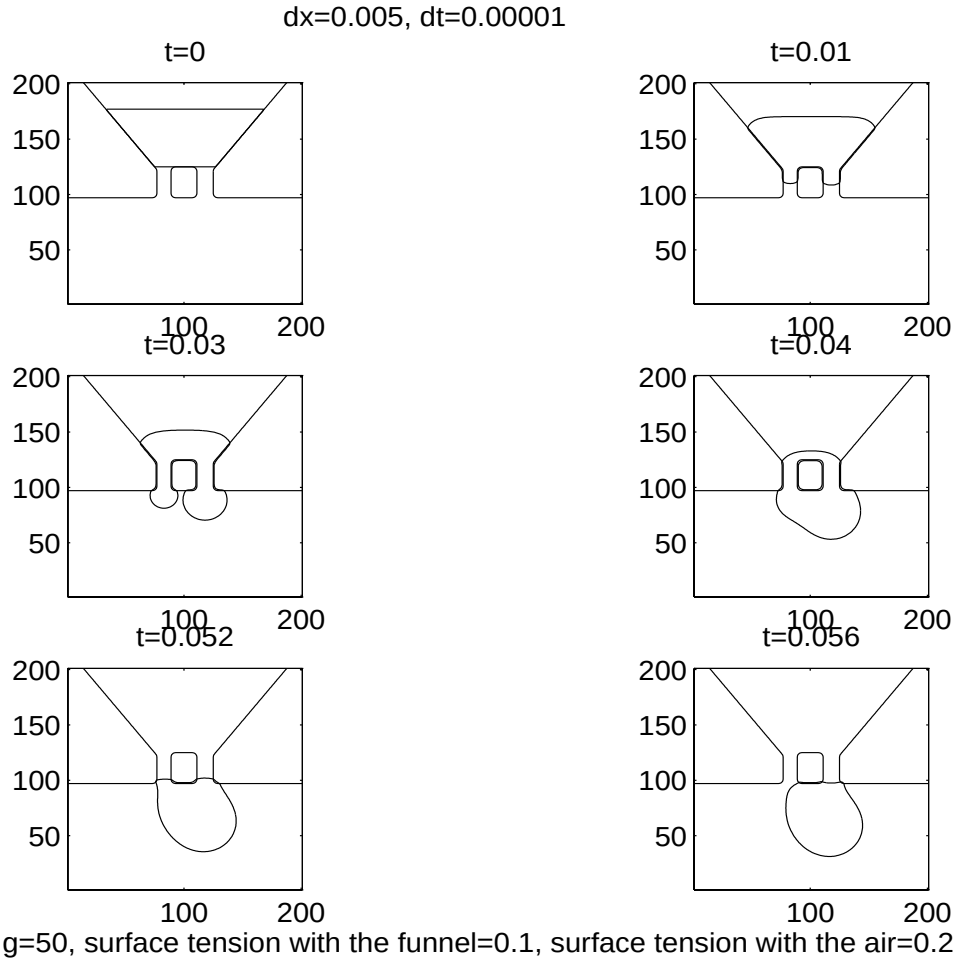


Figure 16: Liquid falling through funnel opening. Reprinted from [90].

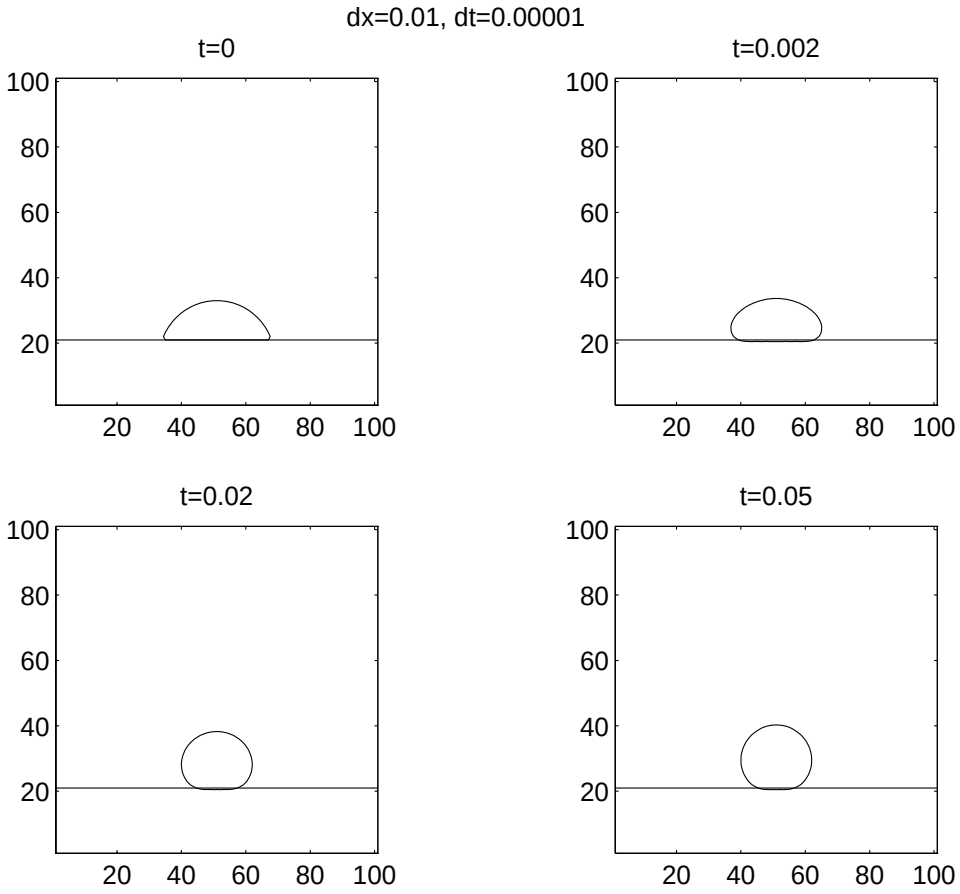


Figure 17: Mercury droplet responding to surface tension. Reprinted from [90].

6 Applications to Computer Vision and Image Processing

The use of PDE's and level set motion in image analysis and computer vision has exploded in recent years. Good references include [18] and [58].

One basic idea is to view an image as $u_0(x, y)$, a function defined on a square, and obtain a (usually second order) flow equation of the form

$$\begin{aligned} u_t &= F(u, Du, D^2u, x, t) \\ u(x, y, 0) &= u_0(x, y) \end{aligned} \tag{27}$$

which, for positive t, processes the image.

For example, if one solves the heat equation with $F(u, Du, D^2u, x, t) = \Delta u$, then $u(x, y, t)$ is the same as convolution of u_0 with a Gaussian of variance t.

L.I. Rudin, in his Ph.D. thesis [70], made the point that images are largely characterized by singularities, edges, boundaries, etc, and thus nonlinearity, especially ideas related to shock propagation, should play a role. This led to the very successful total variation based image restoration algorithms of [72] and [71]. Briefly, if we are presented with a noisy blurred image

$$u_0 = j * u + n \tag{28}$$

where j is a given convolution kernel, and the mean and variance of the noise are given, we wish to obtain the "best" restored image. This leads us (see [72] and [71]) to the evolution equation

$$u_t = \nabla \cdot \frac{\nabla u}{|\nabla u|} - \lambda j * (j * u - u_0) \tag{29}$$

to be solved for $t > 0$, where $u(x, y, 0)$ is given, and $\lambda(t) > 0$ is obtained as a Lagrange multiplier, or is set to be a fixed constant. If $j * u = u$, this becomes a pure denoising problem. The (very interesting) geometric interpretation of this procedure is that each level contour of u is moved normal to itself with velocity equal to its curvature, divided by the norm of the gradient of u, then "pulled back" in an attempt to deconvolve (28). The results are state-of-the-art for many problems. Noisy regions can be thought of as corresponding to contours having very high curvature, while edges have finite curvature and infinite gradients.

Here the motion of level sets is just used to interpret the dynamics. In [4], it was shown that reasonable axioms of image processing lead to the

remarkable fact that motion of level contours by a function of curvature is fundamental to the subject. The artificial time t is actually the scale parameter [4].

We would like to describe a few new applications of this set of ideas. In [10], we have considered the problem of processing of images defined on manifolds. The technique actually can be used to solve a wide class of elliptic equations on manifolds, without triangulation, using only a local Cartesian grid, for very general situations.

Given a manifold in R^3, defined by $\psi(x, y, z) = 0$, we can define the projection matrix

$$P_{\nabla\psi} = I - \frac{\nabla\psi}{|\nabla\psi|} \otimes \frac{\nabla\psi}{|\nabla\psi|}. \tag{30}$$

If u is an image defined on $\psi = 0$ we can use our level set calculus to extend it constant normal to the manifold, in some neighborhood of the manifold.

If u_0 is the original noisy image, the energy to be minimized is

$$E(u) = \int_{R^3} |P_{\nabla\psi}\nabla u|\delta(\psi)|\nabla\psi| d\vec{x} + \frac{\lambda}{2}\int (u - u_0)^2\delta(\psi)|\nabla\psi| d\vec{x}.$$

Using the gradient descent algorithm, i.e. following the general procedure of [72] and [88] leads us to

$$u_t = \frac{1}{|\nabla\psi|}\nabla \cdot \left(\frac{P_{\nabla\psi}\nabla u}{|P_{\nabla\psi}\nabla u|}|\nabla\psi|\right) - \lambda(u - u_0).$$

This corresponds to total variation denoising. This is done using the local level set method [66] which allows great flexibility in geometry, while always using a Cartesian grid. See [10] for denoising and deblurring results.

The technique is quite general – both variational problems and PDE's defined on manifolds can be solved in a reasonably straightforward fashion, without restrictions on the manifold and without complicated triangulation – just by using a fixed Cartesian grid.

Another basic image processing task is to detect objects hidden in an image u_0. A popular technique is called active contours or snakes, in which one evolves a curve, subject to constraints until the curve surrounds the image.

The level set method was first used in [16] as a very convenient tool to follow the motion of active contours in order to surround hidden objects.

This was an important step since topological changes could easily be handled, a variational approach could be easily used [17] and stable, easy to program algorithms resulted.

The curve is moved with a velocity which vanishes when the object is surrounded. Thus edge detectors are traditionally used to stop the evolving curve. For example, one might use

$$g(|\nabla u_0|) = \left(\frac{1}{1+|\nabla j_\sigma * u_0|}\right)^2$$

where j_σ is a Gaussian of variance σ.

In [20] the authors developed a model which was not based on edges, using a scale parameter, based on a simplification of the Mumford-Shah [56] energy based segmentation. The implementation is done through the variational level set approach [88] and the results are remarkable. The method has a denoising capability as well as the ability to perform a multiscale segmentation. See [21] and [20] for details. Here we just present the evolution equation for the level set function φ:

$$\varphi_t = |\nabla\varphi|\left[\mu\nabla\cdot\frac{\nabla\varphi}{|\nabla\varphi|} - \nu - \lambda(u_0-c_1)^2 + \lambda(u_0-c_2)^2\right]$$

for parameters $\mu, \nu, \lambda \geq 0$, where c_1 and c_2 are the averages of u_0 over the region for which $\varphi \geq 0$ and $\varphi \leq 0$ respectively. ν corresponds to the bulk energy of the area for which $\varphi \geq 0$, μ corresponds to the surface tension of the interface, and λ is the penalty for the L^2 error between u_0 and its mean over each region. Figure 18 shows an active contour segmenting a MRI brain image from its backgound.

A somewhat related problem as discussed in [89] is the following. Given a collection of unorganized points, and/or curves, and/or surface patches, find a surface which can be regarded as its shape. This is a fundamental visualization problem which arises in computer graphics, visualization and simulation. No assumptions about the ordering, connectivity or topology of the data sets or of the true shape is given. The input is the general distance to the data set which is given on a (usually logically rectangular) grid. Additionally, we may also input the values of the normal to the surface at the same or different data points.

The key idea is to find a function φ whose zero level set is the interpolating surface, φ changes sign as one goes from inside to outside the surface. The output is the discrete values of φ, which can be reinitialized to be signed distance to this surface.

We set up a variational problem, which basically minimizes the integral over the unknown surface, of the pth power of distance to the data set. We may include information about the normals in analogous fashion.

Gradient descent (as in the image restoration and active contour problems) gives us a weighted motion by curvature plus convection algorithm. The results are very promising as shown figure 19. For more details, see [89].

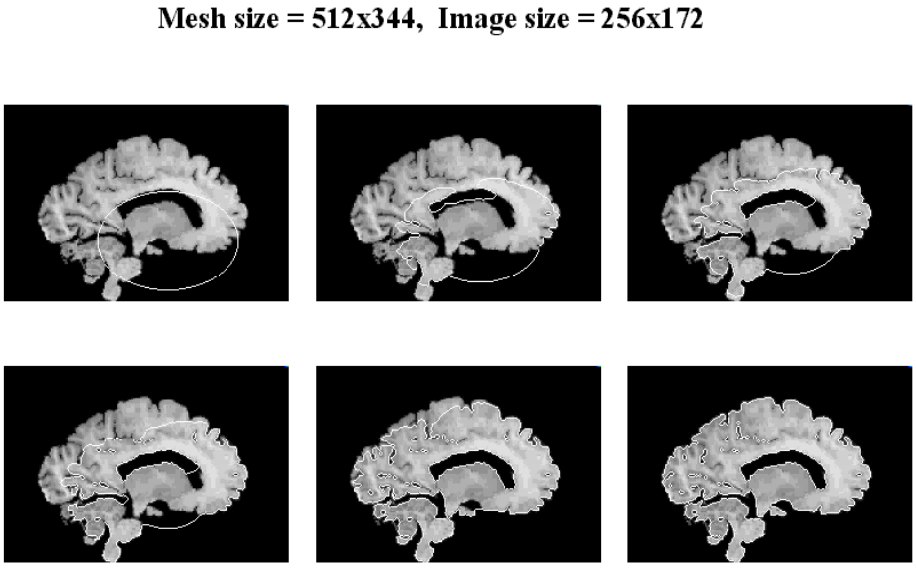


Figure 18: Active contour segmentation of an MRI brain image from its backgound. Reprinted from [19].

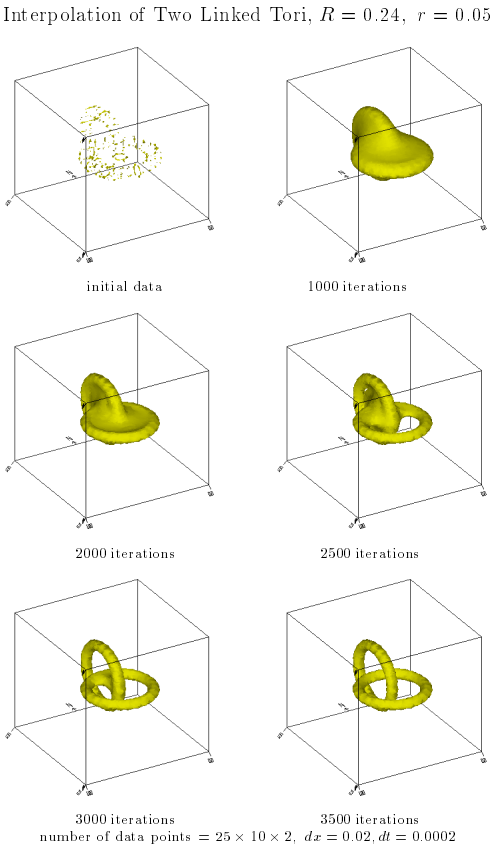


Figure 19: Interpolation of two linked tori. Reprinted from [89].

7 Conclusion

The idea of using a level set to represent an interface is a very old one. The level set method itself has antecedents, for example, in the G equation approach of Markstein [50]. What is new is the level set method technology, theoretical justification through viscosity solutions, and the enormous number of wide ranging applications that are now available, with new applications developing quite frequently.

References

[1] Adalsteinsson, D. and Sethian, J.A., *The Fast Construction of Extension Velocities in Level Set Methods*, J. Comput. Phys. 148, 2-22 (1999).

[2] Adalsteinsson, D. and Sethian, J.A. *A Fast Level Set Method for Propagating Interfaces*, J. Comput. Phys. 118, 269-277 (1995).

[3] Adalsteinsson, D. and Sethian, J.A., *A Level Set Approach to a Unified Model for Etching, Deposition, and Lithography II: Three Dimensional Simulations*, J. Comput. Phys. 122, 348-366 (1995).

[4] Alvarez, L., Guichard, F., Lions, P.-L., and Morel, J.-M., *Axioms and Fundamental Equations of Image Processing*, Arch. Ration. Mech. and Anal. 123, 199-257, (1993).

[5] Ambrosio, L. and Soner, H.M., *Level Set Approach To Mean Curvature Flow in Arbitrary Codimension*, J. Diff. Geom. 43 (4) 693-737 (1996).

[6] Bardi, M. and Evans, L.C., *On Hopf's Formulas for Solutions of Hamilton-Jacobi Equations*, Nonlinear Analysis TMA 8, 1373-1381 (1984).

[7] Bardi, M. and Osher, S., *The Nonconvex Multidimensional Riemann Problem for Hamilton-Jacobi Equations*, SIAM J. on Anal. 22, 344-351 (1991).

[8] Barles, G., *Solutions de Viscosite des Equations de Hamilton-Jacobi*, Springer-Verlag, Berlin (1966).

[9] Bellettini, G., Novaga, M. and Paolini, M., *An Example of Three Dimensional Fattening for Linked Space Curves Evolving by Curvature*, Comm. of Partial Diff. Equations (in press).

[10] Bertalmio, M., Cheng, L.T., Osher, S., and Sapiro, G., *Variational Problems and Partial Differential Equations on Implicit Surfaces: The Framework and Examples in Image Processing and Pattern Formation*, UCLA CAM Report 00-23, J. Comput. Phys. (in review).

[11] Boue, M. and Dupuis, P., *Markov Chain Approximations for Deterministic Control Problems with Affine Dynamics and Quadratic Cost in the Control*, SIAM J. Numer. Anal. 36 (3), 667-695 (1999).

[12] Brackbill, J.U., Kothe, D.B. and Zemach, C., *A Continuum Method for Modeling Surface Tension*, J. Comput. Phys. 100, 335-354 (1992).

[13] Burchard, P., Cheng, L.-T., Merriman, B., and Osher, S., *Motion of Curves in Three Spatial Dimensions Using a Level Set Approach*, UCLA CAM Report 00-29, J. Comput. Phys. (in review).

[14] Burton, W.K., Cabrera, N. and Frank, F.C., *The Growth of Crystals and the Equilibrium Structure of Their Surfaces*, Phil. Trans. Roy. Soc. London, Ser. A, pp. 243-299, (1951).

[15] Caflisch, R.E., Gyure, M., Merriman, B., Osher, S., Ratsch, C., Vvedensky, D. and Zinck, J., *Island Dynamics and the Level Set Method for Epitaxial Growth*, Appl. Math. Lett. 12, 13-22 (1999).

[16] Caselles, V., Catté, F., Coll, T., and Dibo, F., *A Geometric Model for Active Contours in Image Processing*, Numerische Mathematik 66, 1-31 (1993).

[17] Caselles, V., Kimmel, R. and Sapiro, G., *Geodesic Active Contours*, Int. J. Comput. Vision 22, 61-79 (1997).

[18] Caselles, V., Morel, J.-M., Sapiro, G., and Tannenbaum, A. Editors, Special Issue on Partial Differential Equations and Geometry-Driven Diffusion in Image Processing and Analysis, IEEE Transactions on Image Processing, (1998), v. 7, pp. 269-473.

[19] Chan, T., Fedkiw, R., Kang, M. and Vese, L., *Improvements in the Efficiency and Robustness of Active Contour Algorithms*, (in preparation).

[20] Chan, T. and Vese, L., *Active Contours Without Edges*, UCLA CAM Report 98-53 (1998).

[21] Chan, T. and Vese, L., *An Active Contour Model Without Edges*, in Lecture Notes in Comp. Sci., v. 1687, eds. M. Neilsen, P. Johansen, O.F. Olsen and J. Weickert, pp. 141-151, 1999.

[22] Chang, Y.C., Hou, T.Y., Merriman, B. and Osher, S., *A Level Set Formulation of Eulerian Interface Capturing Methods for Incompressible Fluid Flows*, J. Comput. Phys. 124, 449-464 (1996).

[23] Chen, Y.G., Giga, Y. and Goto, S., *Uniqueness and Existence of Viscosity Solutions of Generalized Mean Curvature Flow Equations*, J. Diff. Geom. 33, 749-786 (1991).

[24] Chen, S., Merriman, B., Osher, S. and Smereka, P., *A simple level set method for solving Stefan problems*, J. Comput. Phys. 135, 8-29 (1997).

[25] Cheng, L.T., Fedkiw, R.P., Gibou, F. and Kang, M., *A Symmetric Method for Implicit Time Discretization of the Stefan Problem*, J. Comput. Phys. (in review).

[26] Colella, P., Majda, A., and Roytburd, V., *Theoretical and Numerical Structure for Reacting Shock Waves*, SIAM J. Sci. Stat. Comput. 7 (4), 1059-1080 (1986).

[27] Crandall, M.G., Ishii, H. and Lions, P.-L., *User's Guide to Viscosity Solutions of Second Order Partial Differential Equations*, Amer. Math. Soc. Bull. 27, 1-67 (1992).

[28] DeGiorgi, E., *Barriers, Boundaries, Motion of Manifolds*, Lectures in Pavia, Italy, 1994.

[29] Evans, Y.C. Soner, H.M. and Souganidis, P.E., *Phase Transitions and Generalized Motion by Mean Curvature*, Comm. Pure and Applied Math. 65, 1097-1123 (1992).

[30] Evans, Y.C. and Spruck, J., *Motion of Level Sets by Mean Curvature I*, J. Diff. Geom. 33, 635-681 (1991).

[31] Fedkiw. R.P. *A Symmetric Spatial Discretization for Implicit Time Discretization of Stefan Type Problems*, (unpublished) June 1998.

[32] Fedkiw, R., Aslam, T., Merriman, B., and Osher, S., *A Non-Oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (The Ghost Fluid Method)*, J. Comput. Phys. 152 (2), 457-492 (1999).

[33] Fedkiw, R., Aslam, T., and Xu, S., *The Ghost Fluid Method for Deflagration and Detonation Discontinuities*, J. Comput. Phys. 154 (2), 393-427 (1999).

[34] Fedkiw, R. and Liu, X.-D., *The Ghost Fluid Method for Viscous Flows*, Progress in Numerical Solutions of Partial Differential Equations, Arcachon, France, edited by M. Hafez, July 1998.

[35] Gross, R., *Zur Theorie des Washstrums und Losungsforganges Kristalliner Materie*, Abhandl. Math.-Phys. Klasse Kongl. Sachs, Wiss, 35, pp. 137-202 (1918).

[36] Gyure, M., Ratsch, C., Merriman, B., Caflisch, R.E., Osher, S., Zinck, J. and Vvedensky, D. *Level Set Methods for the Simulation of Epitaxial Phenomena*, Phys. Rev. E 59, R6927-6930 (1998).

[37] Harabetian, E. and Osher, S., *Regularization of Ill-Posed Problems via the Level Set Approach*, SIAM J. Appl. Math. 58, 1689-1706 (1998).

[38] Harabetian, E., Osher, S. and Shu, C.-W., *An Eulerian Approach for Vortex Motion Using a Level Set Approach*, J. Comput. Phys. 127, 15-26 (1996).

[39] Helenbrook, B.T., Martinelli, L. and Law, C.K. *A Numerical Method for Solving Incompressible Flow Problems with a Surface of Discontinuity*, J. Comput. Phys. 148, 366-396 (1999).

[40] Helmsen, J., Puckett, E., Colella, P. and Dorr, M., *Two New Methods for Simulating Photolithography Development in 3D*, Proc. SPIE 2726, 253-261 (1996).

[41] Hou, T., *Numerical Solutions To Free Boundary Problmes*, ACTA Num. 4, 335-416 (1995).

[42] Hou, T., Li, Z., Osher, S. and Zhao, H.-K., *A Hybrid Method for Moving Interface Problems with Application to the Hele-Shaw Flow*, J. Comput. Phys. 134, 236-252 (1997).

[43] Jiang, G.-S. and Peng, D., *Weighted ENO Schemes for Hamilton Jacobi Equations*, SIAM J. Sci. Comput. 21, 2126-2143 (2000).

[44] Kang, M., Fedkiw, R., and Liu, X.-D., *A Boundary Condition Capturing Method for Multiphase Incompressible Flow*, UCLA CAM Report 99-21, J. Comput. Phys. (in review).

[45] Karni, S., *Hybrid multifluid algorithms*, SIAM J. Sci. Comput. 17 (5), 1019-1039 (1996).

[46] Karni, S., *Multicomponent Flow Calculations by a Consistent Primitive Algorithm*, J. Comput. Phys 112, 31-43 (1994).

[47] Kim, Y.-T., Goldenfeld, N. and Dantzig, J., *Computation of Dendritic Microstructures using a Level Set Method*, Physical Review E 62, (2000).

[48] Kobayashi, R., *Modeling and Numerical Simulations of Dendritic Crystal Growth*, Physica D 63, 410 (1993).

[49] Liu, X.-D., Fedkiw, R.P. and Kang, M., *A Boundary Condition Capturing Method for Poisson's Equation on Irregular Domains*, J. Comput. Phys. 160, 151-178 (2000).

[50] Markstein, G.H., *Nonsteady Flame Propagation*, Pergamon Press, Oxford 1964.

[51] Mascarenhas, P., *Diffusion Generated Motion by Mean Curvature*, UCLA CAM Report 92-33, 1992.

[52] Merriman, B., Bence, J. and Osher, S., *Diffusion Generated Motion by Mean Curvature*, in AMS Select Lectures in Math., The Comput. Crystal Grower's Workshop, edited by J. Taylor, AMS Providence, 1993, pp. 73-83.

[53] Merriman, B., Bence, J. and Osher, S., *Motion of Multiple Junctions: A Level Set Approach*, J. Comput. Phys. 112 (2), 334-363 (1994).

[54] Merriman, B., Caflisch, R. and Osher, S., *Level Set Methods with an Application to Modelling the Growth of Thin Films*, in Free Boundary Value Problems, Theory and Applications, pp. 51-70, edited by I. Athanasopoulos, G. Makrikis, and J.F. Rodriguez, CRC Press, Boca Raton, FL 1999.

[55] Mulder, W., Osher, S., and Sethian, J.A., *Computing Interface Motion in Compressible Gas Dynamics*, J. Comput. Phys. 100, 209-228 (1992).

[56] Mumford, D., and Shah, J., *Optimal Approximation by Piecewise Smooth Functions and Associated Variational Problems*, Comm. Pure Appl. Math. 42, 577-685 (1989).

[57] Nguyen, D., Fedkiw, R.P. and Kang, M. *A Boundary Condition Capturing Method for Incompressible Flame Discontinuities*, UCLA CAM Report 00-19, J. Comput. Phys. (in review).

[58] Nielsen, M., Johansen, P., Olsen, O.F., and Weickert, J., editors, *Scale Space Theories in Computer Vision*, in Lecture Notes in Computer Science, v. 1682, Springer-Verlag, Berlin, (1999).

[59] Nochetto, R.H., Paolini, M. and Verdi, C., *An Adaptive Finite Element Method for Two Phase Stefan Problems in Two Space Dimensions, Part II: Implementation and Numerical Experiments*, SIAM J. of Sci. Comput. 12, p. 1207 (1991).

[60] Noh, W.F. and Woodward, P.R., *SLIC (Simple Line Interface Construction)*, in Lecture Notes in Physics, v. 59, edited by A. van de Vooren and P.J. Zandbergen, Springer-Verlag, Berlin (1976), p. 330.

[61] Osher, S. and Helmsen, J., *A Generalized Fast Algorithm with Applications to Ion Etching*, (in progress).

[62] Osher, S. and Merriman, B., *The Wulff Shape as the Asymptotic Limit of a Growing Crystalline Interface*, Asian J. Math. 1 (3), 560-571 (1997).

[63] Osher, S., *A Level Set Formulation for the Solution of the Dirichlet Problem for Hamilton-Jacobi Equations*, SIAM J. on Anal. 24, 1145-1152 (1993).

[64] Osher, S. and Sethian, J.A., *Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations*, J. Comput. Phys. 79, 12-49 (1988).

[65] Osher, S. and Shu, C.W., *High Order Essentially Non-Oscillatory Schemes for Hamilton-Jacobi Equations*, SIAM J. Numer. Anal. 28 (4), 907-922 (1991).

[66] Peng, D., Merriman, B., Osher, S., Zhao, H.-K., and Kang, M., *A PDE-Based Fast Local Level Set Method*, J. Comput. Phys. 155, 410-438 (1999).

[67] Peng, D., Osher, S., Merriman, B. and Zhao, H.-K., *The Geometry of Wulff Crystal Shapes and Its Relations with Riemann Problems*, in Contemporary Mathematics 238, 251-303, edited by G.-Q. Chen and E. DeBenedetto, AMS, Providence, RI, 1999.

[68] Reitich, F. and Soner, H.M., *Three Phase Boundary Motions under Constant Velocities I, the Vanishing Surface Tension Limit*, Proc. Royal Soc. Edinburgh 126A, 837-865 (1996).

[69] Rouy, E. and Tourin, A., *A Viscosity Solutions Approach to Shape-From-Shading*, SIAM J. Num Anal. 29 (3) 867-884 (1992).

[70] Rudin, L.I., *Images, Numerical Analysis of Singularities, and Shock Filters*, Ph.D. thesis, Computer Science Dept., Caltech, #5250:TR:87 (1987).

[71] Rudin, L.I. and Osher, S., *Total Variation Based Restoration with Free Local Constraints*, Proc. ICIP, IEEE Int'l Conf. on Image Processing, Austin, TX, (1994), pp. 31-35.

[72] Rudin, L.I., Osher, S. and Fatemi, E., *Nonlinear Total Variation Based Noise Removal Algorithms*, Physica D 60, 259-268 (1992).

[73] Ruuth, S., Merriman and Osher, S., *A Fixed Grid Method for Capturing the Motion of Self-Intersecting Interfaces and Related PDEs*, UCLA CAM Report 99-22, 1999, J. Comput. Phys. (in review).

[74] Ruuth, S., Merriman, B., Xin, J. and Osher, S., *Diffusion-Generated Motion for Mean Curvature of Filaments*, UCLA CAM Report 98-47, 1998, Comm. Pure and Applied Math (in review).

[75] Sethian, J.A., *Algorithms for Tracking Interfaces in CFD and Materials Science*, Ann. Rev. of Comput. Fluid Mech. (1995).

[76] Sethian, J.A., *Fast Marching Level Set Methods for Three Dimensional Photolithography Development*, Proc. SPIE 2726, 261-272 (1996).

[77] Sethian, J.A., *Fast Marching Methods*, SIAM Review 41, 199-235 (1999).

[78] Sethian, J.A. and Strain, J., *Crystal Growth and Dendritic Solidification*, J. Comput. Phys. 98, 231-253 (1992).

[79] Schwarz, K.W., *Simulations of Dislocations on the Mesoscopic Scale. I. Methods and Examples*, J. Applied Phys. 85, 108-119 (1999).

[80] Schwarz, K.W., *Simulations of Dislocations on the Mesoscopic Scale. II. Application to Strained-Layer Relaxation*, J. Applied Phys. 85, 120-129 (1999).

[81] Soravia, P., *Generalized Motion of a Front Propagating Along Its Normal Direction: A Differential Games Approach*, Nonlinear Analysis TMA 22, 1247-1262 (1994).

[82] Steinhoff, J., Fan, M. and Wang, L., *A New Eulerian Method for the Computation of Propagating Short Acoustic and Electromagnetic Pulses*, J. Comput. Phys. 157, 683-706 (2000).

[83] Sussman, M., Fatemi, E., Smereka, P. and Osher, S., *An Improved Level Set Method for Incompressible Two-Phase Flow*, Computers and Fluids 27, 663-680 (1998).

[84] Sussman, M., Smereka, P. and Osher, S., *A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow*, J. Comput. Phys. 114, 146-159 (1994).

[85] Tsai, R., Zhao, H.-K. and Osher, S. *Fast Sweeping Algorithms for a Class of Hamilton-Jacobi Equations*, (in preparation).

[86] Tsitsiklis, J.N., *Efficient Algorithms for Globally Optimal Trajectories*, IEEE Transactions on Automatic Control 40 (9), 1528-1538 (1995).

[87] Unverdi, S.O. and Tryggvason, G., *A Front-Tracking Method for Viscous, Incompressible, Multi-Fluid Flows*, J. Comput. Phys. 100, 25-37 (1992).

[88] Zhao, H.-K., Chan, T., Merriman, B. and Osher, S., *A Variational Level Set Approach to Multiphase Motion*, J. Comput. Phys. 127, 179-195 (1996).

[89] Zhao, H.-K., Merriman, B., Osher, S. and Kang, M., *Implicit Nonparametric Shape Reconstruction from Unorganized Points using a Variational Level Set Method*, UCLA CAM Report 98-7, 1998, Computer Vision and Image Understanding (to appear).

[90] Zhao, H.-K., Merriman, B., Osher, S. and Wang, L., *Capturing the Behavior of Bubbles and Drops Using the Variational Level Set Approach*, J. Comput. Phys. 143, 495-518 (1998).

Shock Capturing, Level Sets and PDE Based Methods in Computer Vision and Image Processing: A Review on Osher's Contribution

Written on the occasion of Stanley Osher's 60th birthday

Ronald P. Fedkiw[1], Guillermo Sapiro[2] and Chi-Wang Shu[3]

ABSTRACT

In this paper we review the algorithm development and applications in high resolution shock capturing methods, level set methods and PDE based methods in computer vision and image processing. The emphasis is on Stanley Osher's contribution in these areas and in the impact of his work. We will first review the linear stability results for hyperbolic systems. This will be followed by shock capturing methods and we will review the Engquist-Osher scheme, TVD schemes, entropy conditions, ENO and WENO schemes and numerical schemes for Hamilton-Jacobi type equations. Among level set methods we will review implicit surfaces, the setup of level set methods, numerical techniques, fluids and materials, variational approach, high codimension motion, geometric optics, and the computation of discontinuous solutions to Hamilton-Jacobi equations. Among computer vision and image processing we will review the total variation model for image denoising, images on implicit surfaces, and the level set method in image processing and computer vision.

Key Words: shock capturing method, level set method, computer vision, image processing, linear stability.

[1]Department of Computer Science, Stanford University, Stanford, CA 94305, E-mail: fedkiw@cs.stanford.edu.

[2]Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, E-mail: guille@mail.ece.umn.edu.

[3]Division of Applied Mathematics, Brown University, Providence, RI 02912, E-mail: shu@cfm.brown.edu.
Research supported by ARO grant DAAD19-00-1-0405, NSF grants DMS-9804985 and ECS-9906606, NASA Langley grant NCC1-01035 and AFOSR grant F49620-99-1-0077.

1 Introduction

This paper is written on the occasion of Stanley Osher's 60th birthday and serves as a review article on a few selected areas in linear stability, high resolution shock capturing schemes, level set methods, and PDE based methods in computer vision and image processing. The emphasis is on Stanley Osher's contribution in these areas and in the impact of his work.

The study of linear stability for finite difference and other numerical methods for hyperbolic, parabolic, and other types of PDEs is very important. Many important results related to linear stability, especially those for initial-boundary value problems, were obtained in the 60s and 70s. Even today, linear stability results are still crucial for linear and nonlinear problems, for they provide a necessary condition for any scheme to perform nicely.

Shock capturing numerical methods have seen revolutionary developments over the past 20 years. These are methods which deal with the numerical solutions of PDEs with discontinuous solutions. Such PDEs include nonlinear hyperbolic systems such as Euler equations of compressible gas dynamics. The problems are difficult because traditional linear numerical methods are either too diffusive, or give unphysical oscillations near the discontinuities which can lead to nonlinear instabilities. The class of high resolution numerical methods overcomes this difficulty to a large extent.

Level set methods have seen tremendously expanded applications in many areas over the past 15 years. This has been made possible by the flexibility of the level set formulation in dealing with dynamic evolutions and topological changes of curves and surfaces, and by the mathematical theory and numerical tools developed in the past 15 years in studying these methods.

PDE based methods in computer vision and image processing have been actively studied in the past few years. Again, the rapid development of mathematical models, solution tools such as level set methods, and high resolution numerical schemes has made PDE based method one of the major tools in computer vision and image processing.

Stanley Osher has made influential contributions to all these fields. A distinctive feature

of his research is that he emphasizes both fundamental problems in algorithm design and analysis, and practical considerations for the applications of the algorithms. This seems also to be the objective of the Journal of Computational Physics. It is thus not a surprise that a significant portion of Osher's journal publications have appeared in the Journal of Computational Physics. This is particularly the case for Osher's work over the past 15 years. Osher's work has been highly influential according to citation statistics. For example, according to the ISI database, which lists papers in selected journals of high impact since 1975, the 82 papers of Osher listed there have been collectively cited 2,386 times (as of November 20, 2001, the same below). Among these, 11 papers have been cited over 100 times each. The top five highly cited papers are: the paper of Osher and Sethian [147] on level set methods, cited 472 times; the paper of Harten, Engquist, Osher and Chakravarthy [76] on ENO schemes, cited 314 times; the two papers of Shu and Osher [171, 172] on ENO schemes, cited 235 and 231 times respectively; and the paper of Harten and Osher [75] on UNO schemes, cited 189 times. We remark that the top four among these five most highly cited papers of Osher were published in the Journal of Computational Physics. The other papers of Osher having a citation over 100 include: the paper of Osher and Solomon on upwind schemes [149], cited 180 times; the paper of Sussman, Smereka and Osher on level set methods for incompressible two phase flows [177], cited 137 times; the paper of Osher on Riemann solvers and entropy conditions [135], cited 131 times; the paper of Rudin, Osher and Fatemi on total variation based denoising in image processing [159], cited 126 times; the paper of Osher and Chakravarthy on high resolution schemes and the entropy condition [139], cited 108 times, and the paper of Engquist and Osher on a monotone scheme (later referred to as the Engquist-Osher, or EO, scheme in the literature) [42], cited 107 times.

The organization of this paper is as follows. In section 2 we review the earlier work of Osher related to linear stability results. Section 3 is devoted to high resolution shock capturing methods for problems with discontinuous or otherwise nonsmooth solutions. Section 4 contains a review of the very popular level set methods, and finally in section 5 we address

PDE based methods in computer vision and image processing.

2 Linear stability results

The study of linear stability for finite difference and other numerical methods for hyperbolic, parabolic and other types of PDEs is very important. For linear methods approximating smooth solutions, a linear stability analysis (plus some dissipation) is usually enough to guarantee convergence, following the Lax equivalence theorem and Strang's result. Even for nonlinear methods and for methods approximating nonsmooth solutions, linear stability is often an important necessary condition for the algorithms to be useful.

For initial value problems, a von Neumann analysis (via Fourier transform) can be easily performed on a finite difference approximation as a necessary and often also sufficient condition for stability. However, stability for initial-boundary value problems is more difficult to analyze.

Osher's work on linear stability and linear methods was mainly done in the early dates. In [126], following up on a seminal paper of Kreiss [99], Osher used Toeplitz matrices in an elegant way to derive what was later called the GKS condition, i.e., the normal mode condition guaranteeing stability of approximations to initial-boundary value problems for linear hyperbolic equations. This line of work was initiated by Godunov and Ryabenkii [64]. It was made uniform by Kreiss [99], followed up by Osher [126], and generalized by Gustafsson, Kreiss and Sundstrom [70]. In [127] Osher provided more general conditions using similar Toeplitz matrix ideas.

In [128], Osher obtained stability conditions for initial-boundary value problems for parabolic equations, generalizing the work of Varah [190]. References [101, 129, 131] were an attempt to analyze and obtain conditions guaranteeing well posedness of initial boundary value problems for linear hyperbolic equations in regions with corners in the boundaries. Reference [130] showed that the Green's function for the biharmonic equation corresponding to a clamped plate near a right angle corner changes sign an infinite number of times.

In [112], Majda and Osher extended Kreiss' well posedness condition for initial-boundary value problems for hyperbolic equations to those with uniformly characteristic boundaries. In [111], Majda and Osher analyzed the reflection of singularities at the boundary for non-grazing reflection for hyperbolic equations. In [113], Majda and Osher showed how error propagates globally within the domain of dependence for numerical approximations to coupled hyperbolic systems. The paper [110] by Majda, McDonough and Osher was the first to recommend the use of smooth cutoff functions on the frequency domain for spectral methods to confine errors to local regions near propagating discontinuities and for stability. Sharp estimates on the region of propagation were obtained. These cutoffs are now widely used in the literature and the paper is still frequently cited, 45 times total, including many in recent years.

Osher in [132] obtained well-posedness results for linear boundary value problems of mixed elliptic-hyperbolic type; in [33], Deacon and Osher made the method into a finite element approximation for such equations.

In [44], Engquist, Osher and Zhong obtained wavelet based fast algorithms for linear hyperbolic and parabolic equations. Finally, in [41, 50, 49], Engquist, Fatemi and Osher considered numerical methods for high frequency asymptotics for geometric optics. These might be considered nonlinear, since the eikonal equation is.

3 High resolution shock capturing methods

Shock capturing methods refer to a class of numerical methods for solving problems containing discontinuities (shocks, contact discontinuities or other discontinuities), which can automatically "capture" these discontinuities without special effort to track them. A typical situation would be the solution of a hyperbolic conservation law, either a scalar equation or a system, either in one spatial dimension

$$u_t + f(u)_x = 0 \tag{3.1}$$

or in multiple (say, three) spatial dimensions:

$$u_t + f(u)_x + g(u)_y + h(u)_z = 0. \tag{3.2}$$

A main ingredient of shock capturing methods is the conservation form of a scheme, namely, a scheme approximating (3.1) is in the form

$$\frac{du_j}{dt} + \frac{1}{\Delta x}\left(\hat{f}_{j+\frac{1}{2}} - \hat{f}_{j-\frac{1}{2}}\right) = 0 \tag{3.3}$$

where u_j is an approximation to either the point value $u(x_j, t)$ or the cell average $\bar{u}(x_j, t) = \frac{1}{\Delta x}\int_{x_j-\frac{\Delta x}{2}}^{x_j+\frac{\Delta x}{2}} u(x,t)dx$ of the exact solution of (3.1), and $\hat{f}_{j+\frac{1}{2}}$ is a numerical flux which typically depends on a few neighboring points

$$\hat{f}_{j+\frac{1}{2}} = \hat{f}(u_{j-k}, u_{j-k+1}, ..., u_{j+m})$$

and satisfies the following two conditions: it is consistent with the physical flux $f(u)$ in the sense $\hat{f}(u, u, ..., u) = f(u)$, and it is at least Lipschitz continuous with respect to all its arguments. Notice that (3.3) is written in a semi-discrete method of lines form, while in practice the time variable t must also be discretized. Conservative schemes in the form of (3.3) are especially suitable for computing solutions with shocks, because of the important Lax-Wendroff theorem, which states that solutions to such schemes, if convergent, would converge to a weak solution of (3.1). In particular, this means that the computed shocks will propagate with the correct speed. Almost all shock capturing schemes, including those developed by Osher and his collaborators, are of the conservation form (3.3). However, there are certain situations where a relaxation on the strict conservation would be beneficial and would not hurt the convergence to weak solutions under suitable additional assumptions. The work of Osher and Chakravarthy [138] on the "weak conservation form" for schemes on general curvilinear coordinates, and the work of Fedkiw et al. on "ghost fluid" method [56], which treats the fluid interface in a non-conservative fashion, are such examples.

3.1 First Order Monotone Schemes

In the late 70s and early 80s, designing good first order monotone schemes for (3.1) and (3.2), which give monotone shock transitions and can be proven to converge to the physically relevant weak solutions (e.g. Crandall and Majda [32]), was an active research area. The Godunov scheme is a scheme with the least numerical dissipation among first order monotone schemes, however it is costly to evaluate for complex flux functions $f(u)$, and its flux is only Lipschitz continuous but not smoother. The Lax-Friedrichs scheme is easy to evaluate and very smooth but is excessive dissipative.

In [42] and [43], Engquist and Osher designed monotone schemes for the transonic potential equations and for general scalar conservation laws, which are relatively easy to evaluate, are C^1 smooth, and have a small dissipation almost comparable with Gudonov schemes. The main idea is to approximate everything by rarefaction waves (multi-valued solutions suitably integrated over for shocks). These Engquist-Osher schemes soon became very popular, especially for implicit type methods and steady state calculations, for which the extra smoothness of the numerical fluxes helped a lot. Similar schemes for Hamilton-Jacobi equations were given by Osher and Sethian [147].

Later, Osher [133] and Osher and Solomon [149] generalized it to systems of conservation laws, obtaining what was later referred to as Osher scheme in the literature. The Osher scheme for systems has a closed form formula (for Euler equations of gas dynamics and many other systems), hence no iterations are needed, unlike the Godunov scheme. It is smoother (C^1) than the Godunov scheme and also has smaller dissipation than the simpler Lax-Friedrichs scheme. Applications of Osher schemes to the Euler equations can be found in Chakravarthy and Osher [21].

In [145], Osher and Sanders designed a conservative procedure to handle locally varying time and space grids for first order monotone schemes, and proved convergence to entropy solutions for such schemes. These ideas have been used later by Berger and Colella on their adaptive methods.

3.2 High Resolution TVD Schemes

First order monotone schemes are certainly nice in their stability and convergence to the correct entropy solutions, however they are too diffusive for most applications. One would need to use many grid points to get a reasonable resolution, which seriously restricts their usefulness for multidimensional simulations.

In the 70s and early and mid 80s, the so-called "high resolution" schemes, i.e. those schemes which are at least second order accurate and are stable when shocks appear, were developed. These started with the earlier work of, e.g., the FCT methods of Boris and Book [10], and the MUSCL schemes of van Leer [189], and moved to Harten's TVD schemes [74]. Osher and his collaborators did extensive research on TVD schemes, and contributed significantly towards the analysis of such methods, during this period. These include the schemes developed and analyzed in [135], [139], [136], and the very high order (measured by truncation errors in smooth, monotone regions) TVD schemes in [140].

3.3 Entropy Conditions

The entropy condition is an important feature for conservation laws. Because weak solutions are not unique, entropy conditions are needed to single out a unique, physically relevant solution. Osher and his collaborators did extensive research on designing and analyzing entropy condition satisfying numerical methods for conservation laws.

In [114], Majda and Osher proved that the traditional second order Lax-Wendroff scheme, although linearly stable, is not L^2 stable when solving nonlinear conservation laws with discontinuous solutions. They then provided a recipe of adding artificial viscosities, such that the scheme maintained second order accuracy yet could be proven convergent to the entropy solution. This scheme is however oscillatory, hence not very practical in applications.

In [135], Osher provided a general framework to study systematically entropy conditions for numerical schemes. This was followed by the work of Osher and Chakravarthy [139] in the study of high resolution schemes and entropy conditions, the work of Osher [136] on

generalized MUSCL schemes, the work of Osher and Tadmor [150] on entropy condition and convergence of high resolution schemes, and the work of Brenier and Osher [11] on entropy condition satisfying "maxmod" second order schemes. Entropy condition satisfying approximations for the full potential equation of transonic flow were given in [142].

3.4 ENO Schemes

In the mid 80s it was realized that TVD schemes, despite their excellent stability and high resolution properties, have serious deficiency in that they degenerate to first order at *smooth* extrema of the solution [139]. Thus, even though TVD schemes can be designed to any order of accuracy, see for example the schemes up to 13th order accurate in [140], practical TVD schemes are referred to as second order schemes since the global L^1 errors of any TVD scheme can only be second order, even for smooth, non-monotone solutions.

In [75], Harten and Osher relaxed the TVD restriction, and replaced it by a UNO restriction, in that the total number of numerical extrema does not increase and their amplitudes could be allowed to increase slightly. The UNO scheme in [75] is uniformly second order accurate including at smooth extrema. However, it was soon realized that the UNO restriction was still too strong and excluded schemes of higher than second order. Thus, the concept of ENO, or essentially non-oscillatory, schemes was first given by Harten, Engquist, Osher and Chakravarthy [76] in 1987. The clever idea is that of an adaptive stencil, which is chosen based on the local smoothness of the solution, measured by the Newton divided differences of the numerical solution. Thus the order of scheme is never reduced, however the local stencil automatically avoids crossing discontinuities. Such schemes allow both the number of numerical extrema and their amplitudes to increase, however such additional oscillations are controlled on the level of truncation errors even if the solution is not smooth. ENO schemes have been extremely successful in applications, because they are simple in concept, allow arbitrary orders of accuracy, and generate sharp, monotone (to the eye) shock transitions together with high order accuracy in smooth regions of the solution including at the extrema.

The original ENO schemes in [76] are in the cell averaged form, namely they are finite volume schemes approximating an integrated version of (3.1). Finite volume schemes have the advantage of easy handling of non-uniform meshes and general geometry in multi-space dimensions, however they are extremely costly in multi-space dimensions, when the order of accuracy is higher than two, because then one cannot confuse cell averages with point values, as they only agree up to second order accuracy, and a complex reconstruction procedure is needed to obtain point values from cell averages for evaluating the numerical fluxes. The cost is also associated with the high order numerical quadratures needed for evaluating the integration of the numerical fluxes along cell boundaries in multi-dimensions. Later, Shu and Osher [171], [172] developed ENO schemes in finite difference using point values of the numerical solution, but still in conservation form (3.3). An important observation made in [171] and [172] is that the numerical flux $\hat{f}_{j+\frac{1}{2}}$ in (3.3) is *not* a high order approximation to the physical flux at $x_{j+\frac{1}{2}}$: the difference between the numerical flux $\hat{f}_{j+\frac{1}{2}}$ and the physical flux $f(u_{j+\frac{1}{2}})$ is $O(\Delta x^2)$. This is a common mistake among practitioners of finite difference schemes. If a high order interpolation on the point values u_j is performed to obtain a high order approximation to $u_{j+\frac{1}{2}}$, and a numerical flux is chosen to approximate $f(u_{j+\frac{1}{2}})$ to a high order accuracy, then the scheme is only second order accurate. Correct choice of the numerical fluxes to obtain arbitrarily high order accuracy is given in [171] and [172]. The approach in [172] is especially simple. A detailed description of the construction and comparison of finite volume and finite difference ENO schemes can be found in the lecture notes [170].

Also in [171], a class of nonlinearly stable high order Runge-Kutta time discretization methods is developed. Termed TVD time discretizations, these Runge-Kutta methods have become very popular and have been used in many schemes. See, e.g. [65] for a review of such methods.

Analysis of ENO schemes was given in Harten et al. [77]. Applications of ENO schemes to two and three dimensional compressible flows, including turbulence and shear flow calcu-

lations, were given in Shu et al. [173]. Triangle based second order non-oscillatory schemes were given in Durlofsky et al. [37]. Non-oscillatory self-similar maximum principle satisfying high order shock capturing schemes were given in Liu and Osher [106]. Efficient characteristic projection in upwind difference schemes was given in Fedkiw et al. [59]. Convex ENO schemes without using field-by-field projection were given in Liu and Osher [107]. Chemically reactive flows were simulated in Ton et al. [182] and in Fedkiw et al. [58].

The popularity of ENO schemes is demonstrated by the citation statistics: among Osher's five mostly highly cited papers mentioned in the introduction, four of them are about ENO schemes, i.e. [76] (cited 314 times); [171] (cited 235 times); [172] (cited 231 times); and [75] (cited 189 times). The top cited paper of Osher, [147] (cited 472 times) is on level set methods but also uses second order ENO schemes for the numerical solutions and is where the construction of ENO schemes for general Hamilton-Jacobi equations began.

3.5 WENO Schemes

An improvement of ENO scheme is the WENO (weighted ENO) scheme, which was first developed by Liu, Osher and Chan [108]. Both ENO and WENO use the idea of adaptive stencils in the reconstruction procedure based on the local smoothness of the numerical solution to automatically achieve high order accuracy and non-oscillatory property near discontinuities. ENO uses just one (optimal in some sense) out of many candidate stencils when doing the reconstruction; while WENO uses a convex combination of all the candidate stencils, each being assigned a nonlinear weight which depends on the local smoothness of the numerical solution based on that stencil. WENO improves upon ENO in robustness, better smoothness of fluxes, better steady state convergence, better provable convergence properties, and more efficiency.

WENO schemes have been further developed later by Jiang and Shu [88] for fifth order accurate finite difference schemes in one and several space dimensions, by Hu and Shu [80] and Shi et al. [168] for third and fourth order accurate finite volume schemes in two space

dimensions using arbitrary triangulations, and by Balsara and Shu [5] on very high order WENO schemes. A detailed description can again be found in the lecture notes [170].

3.6 Hamilton-Jacobi Equations

We will now move to the description of Osher's work in designing schemes for solving Hamilton-Jacobi equations. Further discussions on this topic will also be given in the next section on level set methods.

In [134], Osher gave explicit formulas for solutions to the Riemann problems for non-convex conservation laws and Hamilton-Jacobi equations. These are important for numerical schemes such as Godunov schemes using such Riemann solvers as building blocks.

In [147], Osher and Sethian, in the context of discussing level set methods, provided a first order monotone scheme (an adaptation of the Engquist-Osher scheme [43]) and a second order ENO scheme based on the framework of [171] and [172]. In [148], Osher and Shu developed high order ENO schemes for solving Hamilton-Jacobi equations, using various building blocks including Lax-Friedrichs, local Lax-Friedrichs, and Roe with an entropy fix. In [102], Lafon and Osher developed high order two dimensional triangle based non-oscillatory schemes for solving Hamilton-Jacobi equations. Later, Jiang and Peng [87] designed WENO schemes for solving Hamilton-Jacobi equations on rectangular meshes and Zhang and Shu [200] designed WENO schemes for solving Hamilton-Jacobi equations on arbitrary triangular meshes.

3.7 Additional Topics

Even though it does not exactly fit the title of this section, the work of Lagnado and Osher [103], [104] is worth mentioning. These papers concern solving an inverse problem to compute the volatility in the European options Black-Scholes model, and they were the first to use PDE techniques to solve this inverse problem, via gradient descent and Tychonoff regularization, allowing the volatility, a coefficient in a parabolic equation to be a function of the independent variables, stock price and time. These papers have attracted a lot of

attention after their publication.

Also worth mentioning is the work of Fatemi, Jerome and Osher [51] on using ENO schemes to solve the hydrodynamic models of semiconductor device simulations. This was the first work of using high order shock capturing methods in semiconductor device simulations, and has led to many further developments, e.g. [86] and [20].

4 Level set methods

4.1 Implicit Surfaces

In n dimensions, consider a surface that separates R^n into separate subdomains with nonzero volumes. For $n = 3$ an explicit representation can be quite difficult to discretize. One needs to choose a number of points on the two dimensional surface and record their connectivity. If the surface and its connectivity is known, it is simple to tile the surface with triangles whose vertices lie on the interface and edges indicate connectivity. On the other hand if connectivity is not known, it can be quite difficult to determine, and even some of the most popular algorithms can produce surprisingly inaccurate surface representations, e.g. surfaces with holes. Connectivity can change for dynamic implicit surfaces, i.e. pinching and merging. Here, connectivity is not a one time issue dealt with when constructing an explicit representation of the surface. Instead, it must be resolved over and over again every time pieces of the surface merge together or pinch apart. The "interface surgery" needed for merging and pinching is complex leading to a number of difficulties. One of the nicest properties of implicit surfaces is that connectivity does not need to be determined for the discretization. A uniform Cartesian grid can be used along with straightforward generalizations of the technology from two spatial dimensions. Possibly the most powerful aspect of implicit surfaces is that it is straightforward to go from two spatial dimensions to three (or even more) spatial dimensions.

Implicit surfaces are defined as the zero isocontour of a function $\phi(\vec{x})$. Consequently implicit interface representations include some powerful geometric tools. For example, we

can determine which side of the interface a point is on simply by looking at the local sign of ϕ. That is, $\vec{x}_o$ is inside the interface when $\phi(\vec{x}_o) < 0$, outside the interface when $\phi(\vec{x}_o) > 0$ and on the interface when $\phi(\vec{x}_o) = 0$.

Implicit functions make simple Boolean operations easy to apply. If ϕ_1 and ϕ_2 are two different implicit functions, then $\phi(\vec{x}) = \min(\phi_1(\vec{x}), \phi_2(\vec{x}))$ is the implicit function representing the union of their interior regions. Similarly, $\phi(\vec{x}) = \max(\phi_1(\vec{x}), \phi_2(\vec{x}))$ represents the intersection of the interior regions. The complement of $\phi_1(\vec{x})$ is $\phi(\vec{x}) = -\phi_1(\vec{x})$. Etc.

The gradient of the implicit function is defined as

$$\nabla\phi = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right). \tag{4.1}$$

$\nabla\phi$ is perpendicular to the isocontours of ϕ pointing in the direction of increasing ϕ. Therefore, if $\vec{x}_o$ is a point on the zero isocontour of ϕ, i.e. a point on the interface, then $\nabla\phi$ evaluated at $\vec{x}_o$ is a vector that points in same direction as the local unit (outward) normal $\vec{N}$ to the interface. Thus, the unit (outward) normal is

$$\vec{N} = \frac{\nabla\phi}{|\nabla\phi|} \tag{4.2}$$

for points on the interface. Equation (4.2) can be used to define a function $\vec{N}$ everywhere on the domain embedding the normal in a function $\vec{N}$ that agrees with the normal for points on the interface. The mean curvature of the interface is defined as the divergence of the normal,

$$\kappa = \nabla \cdot \vec{N} \tag{4.3}$$

so that $\kappa > 0$ for convex regions, $\kappa < 0$ for concave regions and $\kappa = 0$ for a plane.

The characteristic function χ^- of the interior region Ω^- is defined as

$$\chi^-(\vec{x}) = \begin{cases} 1 & \text{if} \quad \phi(\vec{x}) \leq 0 \\ 0 & \text{if} \quad \phi(\vec{x}) > 0 \end{cases} \tag{4.4}$$

where we arbitrarily include the boundary with the interior region. The characteristic function, χ^+ of the exterior region Ω^+ is defined similarly as

$$\chi^+(\vec{x}) = \begin{cases} 0 & \text{if} \quad \phi(\vec{x}) \leq 0 \\ 1 & \text{if} \quad \phi(\vec{x}) > 0 \end{cases} \tag{4.5}$$

again including the boundary with the interior region. $\chi^{\pm}$ are functions of a multidimensional variable $\vec{x}$. It is often more convenient to work with functions of the scalar variable ϕ. Thus we define the one dimensional Heaviside function

$$H(\phi) = \begin{cases} 0 & \text{if} \quad \phi \leq 0 \\ 1 & \text{if} \quad \phi > 0 \end{cases} \tag{4.6}$$

where ϕ depends on $\vec{x}$, although it is not necessary to specify this dependence when working with H. Note that $\chi^+(\vec{x}) = H(\phi(\vec{x}))$ and $\chi^-(\vec{x}) = 1 - H(\phi(\vec{x}))$.

The volume integral (area integral in R^2) of a function f over the interior region Ω^- is defined as

$$\int_{\Omega} f(\vec{x})\chi^-(\vec{x})d\vec{x} \tag{4.7}$$

where the region of integration is all of Ω since χ^- prunes out the exterior region Ω^+ automatically. The one dimensional Heaviside function can be used to rewrite this volume integral as

$$\int_{\Omega} f(\vec{x})\left(1 - H\left(\phi(\vec{x})\right)\right) d\vec{x} \tag{4.8}$$

representing the integral of f over the interior region Ω^-. Similarly,

$$\int_{\Omega} f(\vec{x})H\left(\phi(\vec{x})\right)d\vec{x} \tag{4.9}$$

is the integral of f over the exterior region Ω^+.

By definition, the directional derivative of the Heaviside function H in the normal direction $\vec{N}$ is the Dirac delta function

$$\hat{\delta}(\vec{x}) = \nabla H\left(\phi(\vec{x})\right) \cdot \vec{N} \tag{4.10}$$

which is a function of the multidimensional variable $\vec{x}$. Note that this distribution is only nonzero on the interface $\partial\Omega$ where $\phi = 0$. We can rewrite equation (4.10) as

$$\hat{\delta}(\vec{x}) = H'(\phi(\vec{x}))\nabla\phi(\vec{x}) \cdot \frac{\nabla\phi(\vec{x})}{|\nabla\phi(\vec{x})|} = H'(\phi(\vec{x}))|\nabla\phi(\vec{x})| \tag{4.11}$$

using the chain rule to take the gradient of H and the definition of the normal from equation (4.2). In one spatial dimension, the delta function is defined as the derivative of the Heaviside

function

$$\delta(\phi) = H'(\phi) \tag{4.12}$$

with $H(\phi)$ defined in equation (4.6) above. $\delta(\phi)$ is identically zero everywhere except where $\phi = 0$. This allows us to rewrite equation (4.11) as

$$\hat{\delta}(\vec{x}) = \delta(\phi(\vec{x}))|\nabla\phi(\vec{x})| \tag{4.13}$$

using the one dimensional delta function $\delta(\phi)$.

The surface integral (line integral in R^2) of a function f over the boundary $\partial\Omega$ is defined as

$$\int_\Omega f(\vec{x})\hat{\delta}(\vec{x})d\vec{x} \tag{4.14}$$

where the region of integration is all of Ω since $\hat{\delta}$ prunes out everything except $\partial\Omega$ automatically. The one dimensional delta function can be used to rewrite this surface integral as

$$\int_\Omega f(\vec{x})\delta(\phi(\vec{x}))|\nabla\phi(\vec{x})|d\vec{x}. \tag{4.15}$$

Typically, volume integrals are computed by dividing up the interior region, and surface integrals are computed by dividing up the boundary $\partial\Omega$. This requires treating a complex two dimensional surface in three spatial dimensions. By embedding the volume and surface integrals in higher dimensions, equations (4.8), (4.9) and (4.15) avoid the need for identifying inside, outside or boundary regions. Instead the integrals are taken over the entire region Ω.

Consider the surface integral in equation (4.15) where the one dimensional delta function needs to be evaluated. Since $\delta(\phi) = 0$ almost everywhere, i.e. except on the lower dimensional interface which has measure zero, it seems unlikely that any standard numerical approximation based on sampling will give a good approximation to this integral. Thus, we use a first order accurate smeared out approximation of $\delta(\phi)$. First, we define the smeared out Heaviside function

$$H(\phi) = \begin{cases} 0 & \phi < -\epsilon \\ \frac{1}{2} + \frac{\phi}{2\epsilon} + \frac{1}{2\pi}\sin\left(\frac{\pi\phi}{\epsilon}\right) & -\epsilon \le \phi \le \epsilon \\ 1 & \epsilon < \phi \end{cases} \tag{4.16}$$

where ϵ is a tunable parameter that determines the size of the bandwidth of numerical smearing. A typically good value is $\epsilon = 1.5\triangle x$ making the interface width equal to three grid cells when ϕ is normalized to a signed distance function with $|\nabla\phi| = 1$. Then the delta function is defined according to equation (4.12) as the derivative of the Heaviside function

$$\delta(\phi) = \begin{cases} 0 & \phi < -\epsilon \\ \frac{1}{2\epsilon} + \frac{1}{2\epsilon}\cos\left(\frac{\pi\phi}{\epsilon}\right) & -\epsilon \leq \phi \leq \epsilon \\ 0 & \epsilon < \phi \end{cases} \tag{4.17}$$

where ϵ is determined as above. This delta function allows us to evaluate the surface integral in equation (4.15) using a standard sampling technique such as the midpoint rule. Similarly, the smeared out Heaviside function in equation (4.16) aids in the evaluation of the integrals in equations (4.8) and (4.9).

A distance function $d(\vec{x})$ is defined as

$$d(\vec{x}) = \min|\vec{x} - \vec{x}_I| \quad \text{over all} \quad \vec{x}_I \in \partial\Omega \tag{4.18}$$

implying that $d(\vec{x}) = 0$ on the boundary where $\vec{x} \in \partial\Omega$. For a given point $\vec{x}$, suppose that $\vec{x}_C$ is the point on the interface closest to $\vec{x}$. The line segment from $\vec{x}$ to $\vec{x}_C$ is the shortest path from $\vec{x}$ to the interface. In other words, the path from $\vec{x}$ to $\vec{x}_C$ is the path of steepest descent for the function d. Evaluating $-\nabla d$ at any point on the line segment from $\vec{x}$ to $\vec{x}_C$ gives a vector that points from $\vec{x}$ to $\vec{x}_C$. Furthermore, since d is Euclidean distance,

$$|\nabla d| = 1. \tag{4.19}$$

A signed distance function is an implicit function ϕ with $\phi(\vec{x}) = d(\vec{x}) = 0$ for all $\vec{x} \in \partial\Omega$, $\phi(\vec{x}) = -d(\vec{x})$ for all $\vec{x} \in \Omega^-$, and $\phi(\vec{x}) = d(\vec{x})$ for all $\vec{x} \in \Omega^+$. Given a point $\vec{x}$, and using the fact that $\phi(\vec{x})$ is the signed distance to the closest point on the interface, we can write

$$\vec{x}_C = \vec{x} - \phi(\vec{x})\vec{N} \tag{4.20}$$

to calculate the closest point on the interface where $\vec{N}$ is the local unit normal at $\vec{x}$.

4.2 Level Set Methods

Level set methods add dynamics to implicit surfaces. The key idea that started the level set fanfare was the Hamilton-Jacobi approach to numerical solutions of a time dependent equation for a moving implicit surface. This was first done in the seminal work of Osher and Sethian [147].

Suppose that the velocity of each point on the implicit surface is given as $\vec{V}(\vec{x})$. Given this velocity field, $\vec{V} = \langle u, v, w \rangle$, we wish to move all the points on the surface with this velocity. The simplest way to do this is to solve the ordinary differential equation

$$\frac{d\vec{x}}{dt} = \vec{V}(\vec{x}) \tag{4.21}$$

for every point $\vec{x}$ on the front, i.e. for all $\vec{x}$ with $\phi(\vec{x}) = 0$. This is the Lagrangian formulation of the interface evolution equation. Since there is generally an infinite number of points on the front, this means discretizing the front into a finite number of pieces. For example, one could use segments in two spatial dimensions or triangles in three spatial dimensions. This is not so hard to accomplish if the connectivity does not change and the surface elements do not distort too much. Unfortunately, even the most trivial velocity fields can cause large distortion of boundary elements and the accuracy of the method can deteriorate quickly if one does not periodically modify the discretization in order to account for these deformations by smoothing and regularizing inaccurate surface elements.

In order to avoid problems with instabilities, deformation of surface elements and complicated surgical procedures for topological repair of interfaces, Osher and Sethian [147] proposed using the implicit function ϕ both to represent the interface and to evolve the interface. The evolution of the implicit function ϕ is governed by the simple convection equation

$$\phi_t + \vec{V} \cdot \nabla\phi = 0. \tag{4.22}$$

This is an Eulerian formulation of the interface evolution since the interface is captured by the implicit function ϕ as opposed to being tracked by interface elements as is done in a

Lagrangian formulation. Equation (4.22) is sometimes referred to as the level set equation. The velocity field given in equation (4.22) can come from a number of external sources. For example, when the $\phi(\vec{x}) = 0$ isocontour represents the interface between two different fluids, the interface velocity is calculated using the two-phase Navier-Stokes equations.

In general, one does not need to specify tangential components when devising a velocity field. Since $\vec{N}$ and $\nabla\phi$ point in the same direction, $\vec{T} \cdot \nabla\phi = 0$ for any tangent vector $\vec{T}$ implying that the tangential velocity components vanish when plugged into the level set equation. For example, in two spatial dimensions with $\vec{V} = V_n\vec{N} + V_t\vec{T}$, the level set equation

$$\phi_t + \left(V_n\vec{N} + V_t\vec{T}\right) \cdot \nabla\phi = 0 \tag{4.23}$$

is equivalent to

$$\phi_t + V_n\vec{N} \cdot \nabla\phi = 0. \tag{4.24}$$

Furthermore, since

$$\vec{N} \cdot \nabla\phi = \frac{\nabla\phi}{|\nabla\phi|} \cdot \nabla\phi = \frac{|\nabla\phi|^2}{|\nabla\phi|} = |\nabla\phi| \tag{4.25}$$

we can rewrite equation (4.24) as

$$\phi_t + V_n|\nabla\phi| = 0 \tag{4.26}$$

where V_n is the component of velocity in the normal direction (the normal velocity). Equation (4.26) is *also* known as the level set equation. Equation (4.22) tends to be used for externally generated velocity fields while equation (4.26) tends to be used for (internally) self-generated velocity fields.

4.3 Numerical Techniques

This subsection is a natural continuation of the discussion of numerical methods in section 3.

Consider the one dimensional scalar conservation law

$$u_t + f(u)_x = 0 \tag{4.27}$$

where u is the conserved quantity and $f(u)$ is the flux function. A well known system of conservation laws are the Euler equations for inviscid fluid flow dynamics. The Euler equations are rather interesting because the presence of discontinuities forces one to consider weak solutions where the derivatives of solution variables can fail to exist. While a contact discontinuity is essentially linear, the nonlinear nature of a shock wave discontinuity allows it to develop as the solution progresses forward in time even if the data is initially smooth. Another interesting aspect of the Euler equations concerns the uniqueness of the solution. When more than one solution exists, an entropy condition is needed to pick out the physically correct solution. It turns out that the vanishing viscosity solution is the desired physically correct solution. For example, this vanishing viscosity solution admits a physically consistent rarefaction wave as opposed to a physically inadmissible expansion shock.

Now consider the one dimensional Hamilton-Jacobi equation

$$\phi_t + H(\phi_x) = 0 \tag{4.28}$$

which becomes

$$(\phi_x)_t + H(\phi_x)_x = 0 \tag{4.29}$$

after taking a spatial derivative of the entire equation. Setting $u = \phi_x$ in equation (4.28) results in

$$u_t + H(u)_x = 0 \tag{4.30}$$

which is a scalar conservation law. Thus in one spatial dimension, we can draw a direct correspondence between Hamilton-Jacobi equations and conservation laws. The solution u to conservation law is the derivative of a solution ϕ to a Hamilton-Jacobi equation. Conversely, the solution ϕ to a Hamilton-Jacobi equation is the integral of a solution u to a conservation law. This allows us to point out a number of useful facts. For example, since the integral of a discontinuity is a kink (discontinuity in first derivative), solutions to Hamilton-Jacobi equations can develop kinks in the solution even if the data is initially smooth. In addition, solutions to Hamilton-Jacobi equations cannot generally develop a discontinuity (unless the

corresponding conservation law solution develops a delta function). Thus, solutions ϕ to equation (4.28) are typically continuous. Furthermore, since conservation laws can have nonunique solutions, one needs to apply an entropy condition to pick out the "physically" relevant solution to equation (4.28).

Viscosity solutions for Hamilton-Jacobi equations were first proposed by Crandall and Lions [30] in order to pick out the physically relevant solution. In addition, monotone first order accurate numerical methods were first proven to converge by Crandall and Lions in [31]. Later, in [147], Osher and Sethian used the connection between conservation laws and Hamilton-Jacobi equations to construct higher order accurate artifact free numerical methods based in part on new upwind difference schemes. Even though the analogy between conservation laws and Hamilton-Jacobi equations fails in multidimensions, many Hamilton-Jacobi equations can be discretized in a dimension by dimension fashion. This cumulated in [148] where Osher and Shu proposed a general framework for the numerical solution of Hamilton-Jacobi equations using modern methods from the theory of conservation laws and the multidimensional Riemann solver of Bardi and Osher [6]. The framework in [148] allowed one to use Lax-Friedrichs, Roe-Fix or Godunov building blocks to create higher order accurate spatial discretizations using an essentially non-oscillatory (ENO) polynomial reconstruction introduced in [76] by Harten et al. for the numerical solution of conservation laws. The basic idea is to compute numerical flux functions using the smoothest possible polynomial interpolants. The actual numerical implementation of this idea was improved considerably by Shu and Osher in [171] and [172] where the numerical flux functions were constructed directly from a divided difference table of the pointwise data. [171] and [172] addressed higher order accurate Runge-Kutta discretization in time as well.

In [108], Liu et al. pointed out that the ENO philosophy of picking out exactly one of three candidate stencils is overkill in smooth regions where the data is well behaved. They proposed a Weighted ENO (WENO) method that takes a convex combination of the three ENO approximations. Of course, if any of the three approximations interpolate across a

discontinuity, it is given minimal weight in the convex combination in order to minimize its contribution and the resulting errors. Otherwise, in smooth regions of the flow, all three approximations are allowed to make a significant contribution in a way that improves the local accuracy from third order to fourth order accuracy. Later, Jiang and Shu [88] improved the WENO method by choosing the convex combination weights in order to obtain the optimal fifth order accuracy in smooth regions of the flow. In [87], following the work on HJ ENO in [148], Jiang and Peng extended WENO to the Hamilton-Jacobi framework. This Hamilton-Jacobi WENO or HJ WENO scheme turns out to be very useful as it reduces the numerical errors by more than an order of magnitude over the third order accurate HJ ENO scheme for typical applications.

Even with these high order accurate approaches to solving the Hamilton-Jacobi equations, one can obtain surprisingly inaccurate results when the level set function solution becomes too steep or too flat, i.e. discontinuous or poorly conditioned. In [29], Chopp considered an application where certain regions of the flow had level sets piling up on each other increasing the local gradient, and other regions of the flow had level sets that separated from each other flattening out ϕ. In order to reduce the numerical errors caused by both the steeping and flattening effects, [29] introduced the notion that one should reinitialize the level set function periodically throughout the calculation. In [158], Rouy and Tourin proposed a numerical method for the shape from shading problem that was later generalized into the modern day reinitialization equation of Sussman, Smereka and Osher [177], using the fact that $|\nabla d| = 1$, for d the signed or unsigned distance to a given set.

Unfortunately, this straightforward reinitialization routine can be slow, especially if it needs to be done every time step although [177] noted that just a few time iterations are usually needed. In order to obtain reasonable run times, [29] restricted the calculations of the interface motion and the reinitialization to a small band of points near the $\phi = 0$ isocontour. This idea of computing solutions to Hamilton-Jacobi equations local to the interface has been studied further in the more recent work of Adalsteinsson and Sethian [1] and Peng et

al. [153].

Local methods are important for both solving the Hamilton-Jacobi equation and for reinitializing the level sets so that they do not become discontinuous or poorly conditioned. However, at least in the reinitialization case, it is possible to construct an even faster method that only treats each grid point once while sweeping out from the zero isocontour creating a signed distance function. This algorithm was invented by Tsitsiklis in a pair of papers, [185] and [186]. The most novel part of this algorithm is the extension of Dijkstra's algorithm for computing the taxicab metric to an algorithm for computing Euclidean distance. Although this method was originally proposed by Tsitsiklis, it was later rediscovered by the level set community, see for example Sethian [166] and Helmsen et al. [78].

The great success of level set methods can in part be attributed to the role of curvature in regularizing the level set function such that the proper vanishing viscosity solution is obtained. It is much more difficult to obtain vanishing viscosity solutions with Lagrangian methods that faithfully follow the characteristics. For these methods, one usually has to delete (or add) characteristic information by hand when a shock (or rarefaction) is detected. This ability of level set methods to identify and delete merging characteristics is clearly seen in a purely geometrically driven flow where a square is advected inward normal to itself at constant speed. In the corners of the square, the flow field has merging characteristics that are appropriately deleted by the level set method. On the other hand, repeating the same calculation with a Lagrangian numerical method is difficult since characteristics will merge in the corners of the square but not be automatically deleted. One does not easily obtain the correct viscosity solution. Level set methods are not perfect however, since they tend to incorrectly delete characteristics in under resolved regions of the flow – a behavior frequently called "loss of mass" (or volume) in reference to the error it represents when level sets are used to model incompressible fluid flow. In contrast, despite a lack of explicit enforcement of mass (or volume) conservation, Lagrangian schemes are quite successful in conserving mass since they preserve material characteristics for all time, i.e. characteristics are never deleted.

The difficulty stems from the fact that the level set method cannot accurately tell if characteristics merge, separate, or run parallel in under-resolved regions of the flow. This indeterminacy leads to vanishing viscosity solutions that can incorrectly delete characteristics when they appear to be merging. In [46], Enright et al. designed a hybrid particle level set method to alleviate the mass loss issues associated with level set methods. In the case of fluid flows, knowing a priori that there are no shocks present in the fluid velocity field, one can assert that characteristic information associated with that characteristic field should never be deleted. Particles are randomly seeded near the interface and passively advected with the flow. When marker particles cross over the interface, it indicates that characteristic information has been incorrectly deleted, and these errors are fixed by locally rebuilding the level set function using the characteristic information present in these escaped marker particles.

4.4 Fluids and Materials

Chronologically, the first attempt to use the level set method for flows involving external physics was in the area of two phase inviscid compressible flow. Mulder et al. [122] appended the level set equation to the standard equations for one phase compressible flow. The level set was advected using the velocity of the compressible flow field so that the zero level set of ϕ corresponds to particle velocities and can be used to track an interface separating two different compressible fluids. Later, Karni [90] pointed out that such method suffered from spurious oscillations at the interface. This was later fixed by Fedkiw et al. [56] by creating a set of fictitious ghost cells on each side of the interface, and populating these ghost cells with a specially chosen ghost fluid that implicitly captures the Rankine-Hugoniot jump conditions across the interface. This method was referred to as the ghost fluid method. Later, Fedkiw et al. [57] extended the level set method and ghost fluid method to treat shock, detonation and deflagration waves as sharp discontinuities. Caiden et al. [15] extended these methods to couple incompressible flows to compressible flows in order to study liquid/gas interac-

tions. Fedkiw [55] extended these methods to couple Lagrangian calculations to Eulerian calculations in order to study solid/fluid interactions.

The earliest real success in the coupling of the level set method to problems involving external physics came in computing two-phase incompressible flow, in particular see Sussman et al. [177] and Chang et al. [24]. The Navier-Stokes equations were used to model the fluids on both sides of the interface. Generally, the fluids will have different densities and viscosities and the presence of surface tension forces cause the pressure to be discontinuous across the interface as well. Although these early papers smeared out these discontinuous quantities across the interface, this was later remedied by Kang et al. [89] using the methods developed by Liu et al. [109]. More recently, Nguyen et al. [124] extended these techniques to treat low speed flames.

A level set regularization procedure was proposed in Harabetian and Osher [72] for ill-posed problems such as vortex motion in incompressible flows. This regularization, coupled with non-oscillatory numerical methods for the resulting level set equations, provides a regularization which is topological and is automatically accomplished through the use of numerical schemes whose viscosity shrinks to zero with grid size. There is no need for explicit filtering, even when singularities appear in the solution. The method also has the advantage of automatically allowing topological changes such as merging of surfaces.

An application of this procedure for incompressible vortex motion was given in Harabetian, Osher and Shu [73]. An Eulerian, fixed grid, approach to solve the motion of an incompressible fluid, in two and three dimensions, in which the vorticity is concentrated on a lower dimensional set, is provided. The numerical variables for the level sets are actually smooth, thus allowing for accurate numerical simulations. Numerical examples including two and three dimensional vortex sheets, two dimensional vortex dipole sheets and point vortices, are given. This was the first three dimensional vortex sheet calculation in which the sheet evolution feeds back to the calculation of the fluid velocity, although vortex in cell calculations for three dimensional vortex sheets were done earlier by Trygvasson et al. in

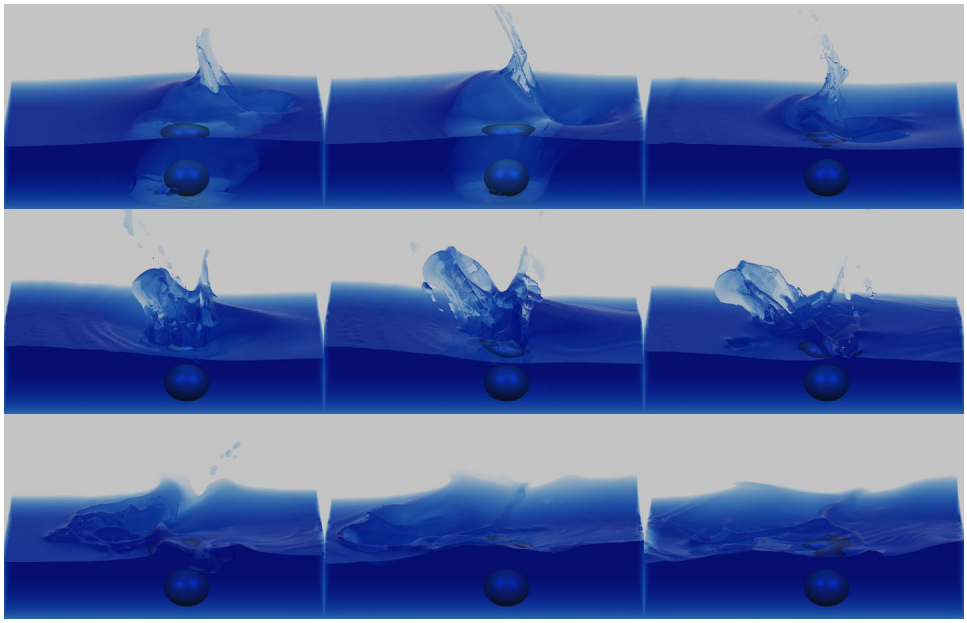


Figure 4.1: A splash is generated as a sphere is thrown into the water.

[183].

Level set methods have been applied to a variety of other problems as well. They have been used to compute solutions to Stefan problems to study crystal growth [26, 94], to simulate water for computer graphics applications [60] as shown in Figures 4.1 and 4.2, and to reconstruct three dimensional models from arbitrary unorganized data points [202] as shown in Figures 4.3 (before) and 4.4 (after).

Level set type analysis was also used to obtain rigorous results identifying the Wulff minimizing shape and the evolution of growing crystals moving with normal velocity defined as a given positive function of the normal direction, thus verifying a conjecture of Gross. Moreover it was also shown that the Wulff energy decreases monotonically under such an evolution to its minimum [143]. A spinoff came in [152] where it was proven that any two dimensional Wulff shape can be interpreted as the solution a corresponding Riemann problem for a scalar conservation law – jumps in the direction of the normal correspond to contact discontinuities, smoothly varying thin flat faces correspond to rarefaction curves and planar facets correspond to constant states. The work in [143] also motivated the derivation of a

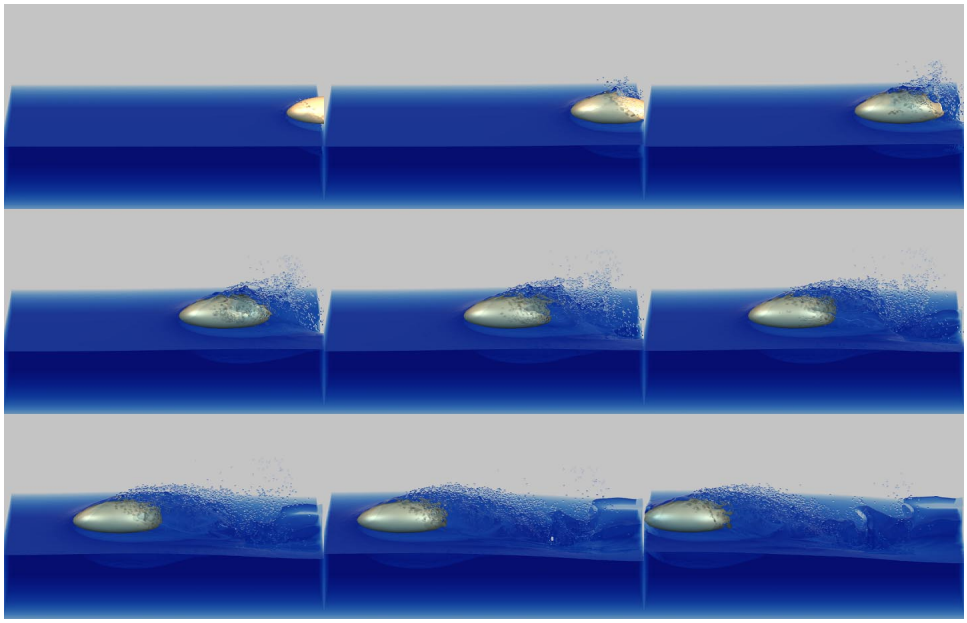


Figure 4.2: An interesting spray effect is generated as a slightly submerged ellipse slips through the water.

new class of isoperimetric inequalities for convex plane curves [67].

Molecular beam epitaxy (MBE) is a method for growing extremely thin films of material. A new continuum model for the epitaxial growth of thin films has been developed. This new island dynamics model has been designed to capture the larger length scale features. The key idea involves the level set based motion of islands of various integer levels – see for example [121, 25, 71].

4.5 A Variational Approach

In [201] a variational level set approach was developed. Key ideas were the use of a single level set function for each phase, the gradient projection method of [159] to prevent overlap and / or vacuum, and the liberal use of the level set calculus as described earlier. This general variational approach has many applications. The first was to study the behavior of bubbles and droplets in two and three dimensions [203], for example drops falling or remaining attached to a generally irregular ceiling, and mercury sitting on the floor.

Many problems in engineering design involve optimizing the geometry to maximize a

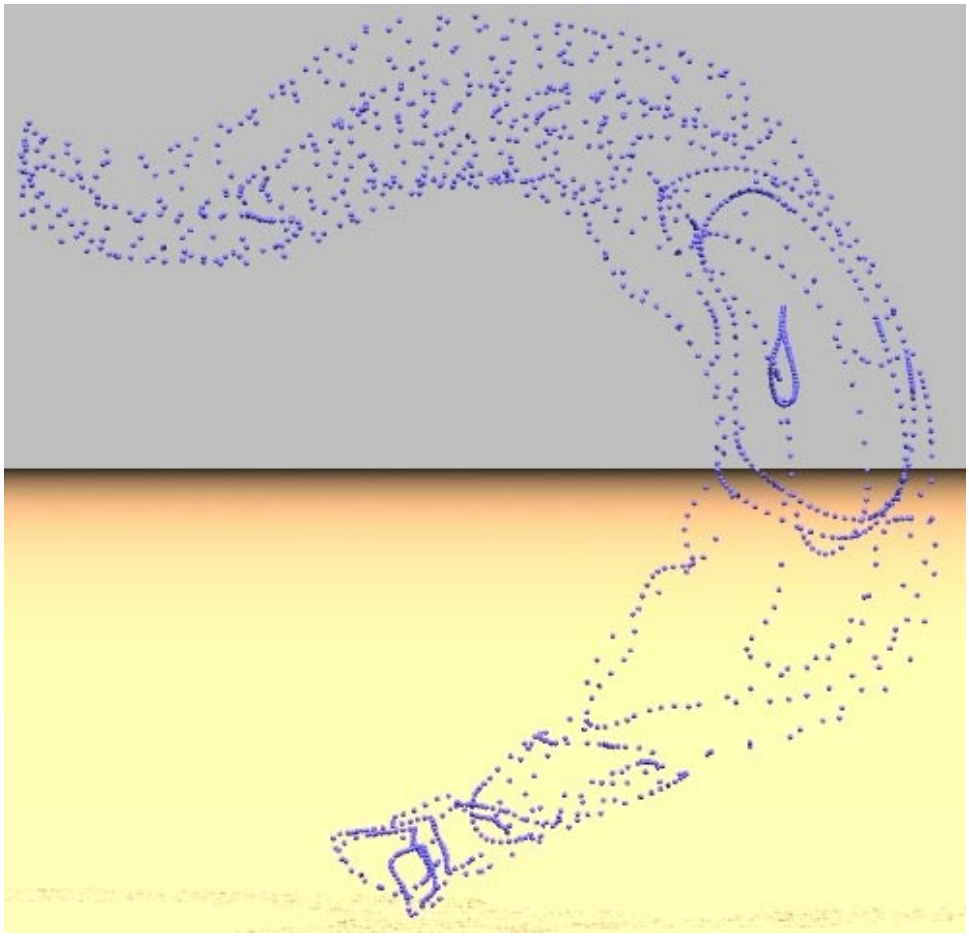


Figure 4.3: Arbitrary data points measured from a rat brain.

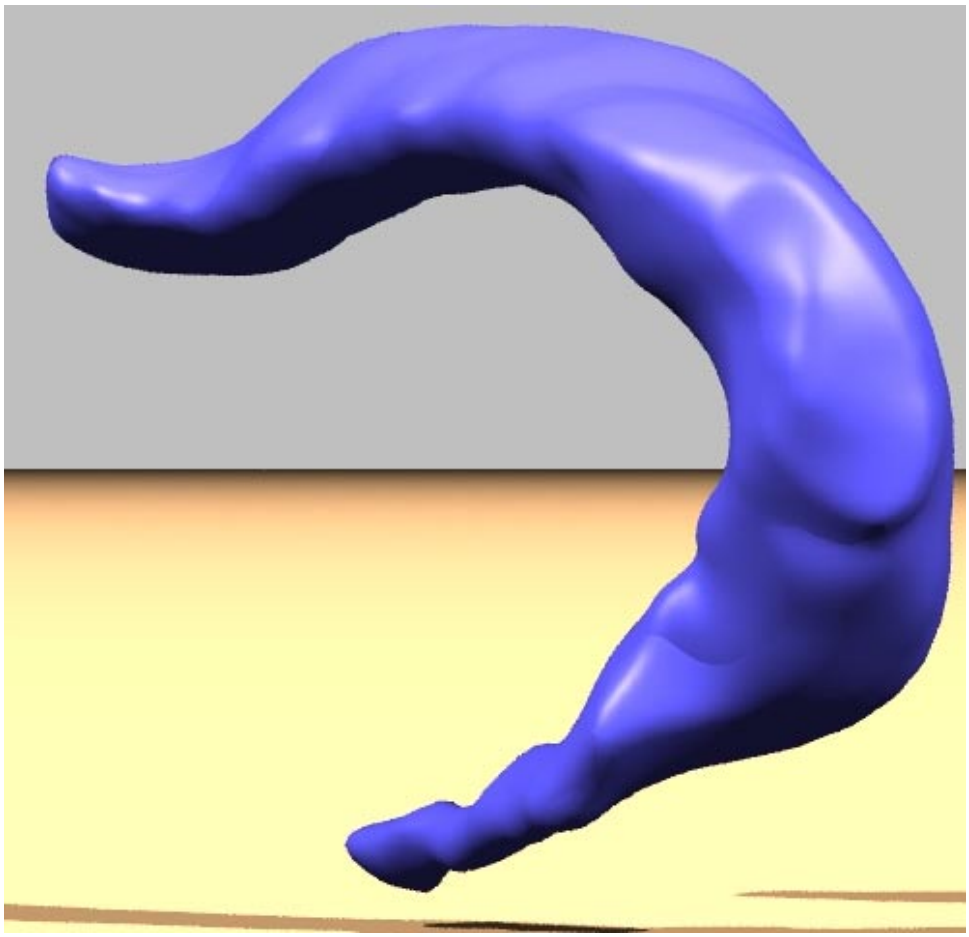


Figure 4.4: A three dimensional geometric reconstruction of the rat brain using the level set method.

certain design objective. In [146] the variational level set method was used to analyze a vibrating system whose resonant frequency or whose spectral gap is to be optimized subject to constraints on the geometry.

This variational approach has applications in computer vision as well, e.g. snakes and active contours [22]. This will be discussed further in section 5.

4.6 High Codimension Motion

Typically level set methods are used to model codimension one objects, e.g. curves in R^2 or surfaces in R^3. In [13], this technology was extended to treat codimension two objects, e.g. curves in R^3, using the intersection of the zero level sets of two functions. This means a curve is determined by

$$\Gamma(t) = \{\vec{x} | \phi_1(\vec{x}, t) = \phi_2(\vec{x}, t) = 0\}.$$

The geometry of the curve can be derived from ϕ_1 and ϕ_2. For example, the tangent to the curve is defined by

$$\vec{T} = \frac{\nabla\phi_1 \times \nabla\phi_2}{|\nabla\phi_1 \times \nabla\phi_2|}.$$

The curvature times the normal is the derivative of the tangent vector along the curve,

$$\kappa\vec{N} = \nabla\vec{T} \cdot \vec{T}. \tag{4.31}$$

The normal vectors can be defined by normalizing this quantity,

$$\vec{N} = \frac{\kappa\vec{N}}{|\kappa\vec{N}|}. \tag{4.32}$$

The binormal is

$$\vec{B} = \frac{\vec{T} \times \vec{N}}{|\vec{T} \times \vec{N}|}.$$

The torsion times the normal vector is $\tau\vec{N} = -\nabla\vec{B} \cdot \vec{T}$.

These geometric quantities are all defined numerically just as in the standard codimension one level set method. Geometric motion of a curve in R^3 is thus obtained by solving coupled

systems of two evolution equations. This is done locally near $\Gamma(t)$, saving on storage and complexity. See [13] for results involving merging and breaking which appear to agree with the reaction-diffusion limit when appropriate.

Another application of this idea comes from the following observation. If we freeze one of the functions, say ϕ_1, we can generate the motion of curves on a surface. Here the surface is defined by $\{\vec{x}|\phi_1(\vec{x}) = 0\}$ and the evolving curve is defined by the intersection of that fixed surface with $\{\vec{x}|\phi_2(\vec{x}, t) = 0\}$. This is useful for path planning on terrain data, see [28].

4.7 Geometric Optics

In [141] a level set based approach for ray tracing and for the construction of wavefronts in geometric optics was introduced. The approach automatically handles the multivalued solutions that appear and automatically resolves the wavefronts. The key idea, first introduced in [45] in a "segment projection" (rather than a level set) approach, is to use the linear Liouville equation in twice as many independent variables and solve in this higher dimensional space via the idea introduced in [13].

In two dimensional ray tracing, this involves solving for an evolving curve in x, y, θ space, where θ is the angle of the normal to the curve. This uses two level set functions and gives codimension 2 motion in 3 space dimension plus time. A local level set method can be used to make the complexity tractable – $O(n^2 \log(n))$ – for n the number of points on the curve for every time iteration. The memory requirement is $O(n^2)$.

In three dimensional ray tracing, this involves solving for an evolving two dimensional surface in x, y, z, θ, ψ space, where θ and ψ give the angle of the normal, and results in codimension 3 motion in 5 space dimension plus time. The complexity goes up by a power of n over the two dimensional case, as does the memory requirement. Again, this involves a local level set method, this time using three level set functions. The interested reader is referred to [175] and [162] for a different Eulerian approach.

4.8 Computing Discontinuous Solutions to Hamilton-Jacobi Equations

Hamilton-Jacobi equations of the form

$$\phi_t + H(\vec{x}, t, \phi, \nabla\phi) = 0 \tag{4.33}$$

have uniformly continuous solutions if H is non-decreasing in ϕ. However, there are interesting cases in which this hypothesis fails. Moreover, discontinuous initial data is appropriate for some problems in control theory and differential games. The solution devised in [63] uses the evolution of the level set of an auxiliary level set equation. The idea has antecedents in [137] where it was proven that, under reasonable circumstances, the zero level set of the viscosity solution of

$$\phi_t + H(\vec{x}, \nabla\phi) = 0$$

for H homogeneous of degree one in $\nabla\phi$ is the same as the t level set of the viscosity solution of

$$H(\vec{x}, \nabla\psi) = 1$$

i.e.

$$\{\vec{x}|\phi(\vec{x}, t) = 0\} = \{\vec{x}|\psi(\vec{x}) = t\}. \tag{4.34}$$

This idea was used in [63] to go one dimension higher in equation (4.33). This leads to new and successful numerical methods for a wide class of initial value problems for Hamilton-Jacobi equations with discontinuous solutions, see [184].

5 Image processing and computer vision

The use of partial differential equations (PDE's) and curvature driven flows in image processing and computer vision has become an active research topic in the past few years. The basic idea is to deform a given curve, surface, or image with a PDE, and obtain the desired result as the solution of this PDE. Sometimes, as in the case of color images, a system of coupled

PDE's is used. The art behind this technique is in the design, analysis, and implementation of these PDE's.

Partial differential equations can be obtained from variational problems. Assume a variational approach to an image processing problem formulated as

$$\arg\left\{\mathrm{Min}_u \mathcal{U}(u)\right\},$$

where $\mathcal{U}$ is a given energy computed over the image (or surface) u. Let $\mathcal{F}(\cdot)$ denote the Euler derivative (first variation) of $\mathcal{U}$. Since under general assumptions, a necessary condition for u to be a minimizer of $\mathcal{U}$ is that $\mathcal{F}(u) = 0$, the (local) minima may be computed via the steady state solution of the equation

$$\frac{\partial u}{\partial t} = -\mathcal{F}(u),$$

where t is an 'artificial' time marching parameter. PDE's obtained in this way have been used already for quite some time in computer vision and image processing, and the literature is large. The most classical example is the Dirichlet integral,

$$\mathcal{U}(u) = \int |\nabla u|^2(x)dx,$$

which is associated with the linear heat equation

$$\frac{\partial u}{\partial t}(x,t) = \Delta u(x).$$

Extensive research is also being done on the direct derivation of evolution equations which are not necessarily obtained from the energy approaches. The attributes of PDE's in image processing are discussed for example in [19, 163]. In the pioneering paper [2] the authors prove that a few basic image processing principles naturally lead to PDE's.

Note that when considering PDE's for image processing and numerical implementations, we are dealing with derivatives of non-smooth signals, and the right framework must be defined. As introduced by the image processing group formerly at CEREMADE [2, 3], the theory of *viscosity solutions* provides a framework for rigorously employing a partial

differential formalism, in spite of the fact that the image may not be smooth enough to give a classical sense to derivatives involved in the PDE. These works also showed with a very elegant axiomatic approach the importance of PDE's in image processing.

Ideas on the use of PDE's in image processing go back at least to Gabor [62] and to Jain [85]. The field took off thanks to the independent works of Koenderink [98] and Witkin [195]. These researchers rigorously introduced the notion of *scale-space*, that is, the representation of images simultaneously at multiple scales. In their work, the multi-scale image representation is obtained by Gaussian filtering, see below. This is equivalent to deforming the original image via the classical heat equation, obtaining in this way an isotropic diffusion flow. In the late 80's, R. Hummel [82] noted that the heat flow is not the only parabolic PDE that can be used to create a scale-space, and indeed argued that an evolution equation which satisfies the maximum principle will define a scale-space as well. The maximum principle appears to be a natural mathematical translation of *causality*. Koenderink once again made a major contribution into the PDE's arena (this time probably involuntarily, since the consequences were not clear at all in his original formulation), when he suggested to add a thresholding operation to the process of Gaussian filtering. As later suggested in [119, 120, 161], and proved by a number of groups [4, 47, 83, 84], this leads to a geometric PDE, actually, one of the most famous ones, curvature motion. In [160] this was extended to diffusion generated motion of curves in $\mathbb{R}^3$. Solving a vector heat equation and thresholding leads to moving the curve in the direction of the normal with velocity equal to its curvature.

Perona and Malik's work [154] on anisotropic diffusion, together with the work by Rudin-Osher-Fatemi on Total Variation [159] (and Osher-Rudin on shock filters [144]), have been among the most influential papers in the area, explicitly showing the importance of understanding non-linear PDE's theory to deal with images. They proposed to replace the linear Gaussian smoothing, equivalent to isotropic diffusion via the heat flow, by a selective non-linear diffusion that preserves edges, see below. Their work opened a number of theoretical and practical questions that continue to occupy the PDE image processing community, e.g.,

[3, 157]. We should also point out that about at the same time, Price *et al.* published a very interesting paper on the use of Turing's reaction-diffusion theory for a number of image processing problems [156]. Reaction diffusion equations were also suggested to create artificial texture [188, 197].

Many of the PDE's used in image processing and computer vision are based on moving curves and surfaces with curvature based velocities. In this area, the level-set numerical method developed by Osher and Sethian [147] is very influential and examples will be provided later in this section. The representation of static objects as level-sets (zero-sets) is of course not completely new to the computer vision and image processing communities, since it is one of the fundamental techniques in mathematical morphology [164]. Considering the image itself as a collection of its level-sets, and not just as the level-set of a higher dimensional function, is a key concept in the PDE's community [2]. Implicit surfaces and level-set representations appear in computer graphics as well [9, 196].

Other works, like the segmentation approach of Mumford and Shah [123] and the snakes of Kass, Witkin, and Terzopoulos [91] have been very influential in the PDE's community as well. More on this will be mentioned below.

It should be noted that a number of the above approaches rely quite heavily on a large number of mathematical advances in differential geometry for curve evolution [66] and in viscosity solutions theory for curvature motion (see e.g., [27, 48].)

The frameworks of PDE's and geometry driven diffusion have been applied to many problems in image processing and computer vision, since the seminal works mentioned above. Examples include continuous mathematical morphology, invariant shape analysis, shape from shading, segmentation, tracking, object detection, optical flow, stereo, image denoising, image sharpening, contrast enhancement, and image quantization. In this section we provide a few examples of these. Since this is a paper in honor of Stan Osher, the presentation of the examples is of course biased by his involvement and contributions in the area. Important sources of literature in the area are the excellent collection of papers in the book edited

by Bart Romeny [157], the book by Guichard and Morel [69] that contains an outstanding description of the topic from the point of view of iterated infinitesimal filters, Sethian's book on level-sets [165], Osher-Fedkiw's long expected book, Lindeberg's book, a classic in Scale-Space theory [105], Weickert's book on anisotropic diffusion in image processing [192], Kimmel's lecture notes [97], Sapiro's recent book [163], Toga's book on Brain Warping that includes a number of PDE's based algorithms for this [181], the special issue on the March 1998 issue of the IEEE Transactions on Image Processing, the special issues in the Journal of Visual Communication and Image Representation, a series of Special Sessions at a number of IEEE International Conference on Image Processing (ICIP), the Proceedings of the Scale Space Workshops, and the 2001 Workshop on Level-Set and Variational Methods. The interested reader will find in these publications some fascinating contributions in the area of PDE's in image processing and computer vision, much beyond the few introductory examples provided below.

5.1 The Total Variation Model for Image Denoising

As mentioned above, the use of PDE's for image enhancement has become one of the most active research areas in image processing [19]. In particular, diffusion equations are commonly used for image regularization, denoising, and multiscale representations (representing the image simultaneously at several scales or levels of resolution). This started with the pioneering works in [98, 195], where the authors suggested the use of the linear heat flow for this task, given by

$$\frac{\partial u}{\partial t} = \Delta u, \tag{5.1}$$

where $u : \Omega \subset I\!R^2 \rightarrow I\!R$ represents the image gray values (the original noisy image is used as initial condition). As it is well known, this equation is the gradient-descent of

$$\int_\Omega \| \nabla u \|^2 \, d\Omega, \tag{5.2}$$

An example of the effect of the linear heat flow or Laplace equation (5.1) is presented in Figure 5.1. It is clear that although this technique can be used to denoise images, it is also

blurring them. That is, not only the noise is being removed, but the edges and the relevant information is getting destroyed as well. Moreover, it can be shown that edges are destroyed faster than the actual noise is removed [8].

Figure 5.1: Example of the heat flow (isotropic diffusion). On the left we have the original image and on the right two different time steps of the diffusion flow, showing how the image is getting blurred.

Two directions were taken to address this problem. On one hand, Perona and Malik [154] suggested to replace the linear heat flow by a PDE that preserves edges. Simultaneously, Rudin, Osher, and Fatemi [159] started to look at the modification of the variational problem (5.2). In certain cases, the two directions can be shown to be equivalent, the PDE being the gradient descent of the proposed variational formulation. Rudin, Osher and Fatemi suggested to replace the linear L_2 norm in (5.2) by the edge oriented Total Variation (TV) norm in the energy, thereby obtaining

$$\int_{\Omega} \parallel \nabla u \parallel d\Omega, \tag{5.3}$$

whose gradient descent flow is given by

$$\frac{\partial u}{\partial t} = \operatorname{div}\left(\frac{\nabla u}{\parallel \nabla u \parallel}\right). \tag{5.4}$$

We notice that in comparison with the linear heat flow, the TV one has a stopping term of the form $\frac{1}{\|\nabla u\|}$. This helps to preserve edges, as can be seen in Figure 5.2. Rudin *et al* also suggested to add constraints to this minimization, in order to avoid reaching the trivial (flat) steady state, thereby improving the results in Figure 5.2. In this case the corresponding

Lagrange multiplier is evaluated via a projection method that was found to be useful in other applications as well, e.g., [146].

Figure 5.2: Example of the TV flow for anisotropic diffusion. On the left we have the original image and on the right the result of the flow, showing how the edges are much better preserved than with the isotropic flow.

From the point of view of edge preservation, the TV flow is optimal if we limit ourselves to convex functionals [8]. Motivated by the seminal work of Perona and Malik and Rudin-Osher-Fatemi, significant theoretical and practical studies have been conducted in this kind of anisotropic diffusion flows in general and the TV flow in particular. People have studied their numerics (e.g., Mulet-Chan-Golub, Weickert, Marquina-Osher) as well as their formal mathematical properties (e.g., Alvarez-Morel-Lions, Weickert, to name just a few, and more recently Caselles *et al.* with a full study of the TV flow in general dimensions). This work has also in part motivated Cohen, DeVore, and others to connect wavelets with the TV space.

5.2 Images on Implicit Surfaces

In the last section we dealt with images on the plane. There is of course more than that, and data can be defined on surfaces. In [7] the authors dealt with this issue. A framework for solving variational problems and partial differential equations for scalar and vector-valued data defined on surfaces was introduced. The key idea is to implicitly represent the static surface as the level set of a higher dimensional static function, and solve the surface equations

in a fixed Cartesian coordinate system using this new embedding function. Implicit surfaces can be obtained for example from the algorithms in [39, 61, 100, 117, 179, 186, 199, 202]. Applications of PDE's on surfaces include computer graphics [187, 188, 197], visualization [35], weathering simulation [36], vector field computation or interpolation process [155, 194], inverse problems [53], and surface parameterization [38].

We assume then that the three dimensional surface $\mathcal{S}$ of interest is given in implicit form, as the zero level set of a given function $\phi : I\!R^3 \rightarrow I\!R$. This function is negative inside the closed bounded region defined by $\mathcal{S}$, positive outside, Lipschitz continuous a.e., with $\mathcal{S} \equiv \{x \in I\!R^3 : \phi(x) = 0\}$. To ensure that the data, which needs not to be defined outside of the surface originally, is now defined in the whole band, one simple possibility is to extend this data u defined on $\mathcal{S}$ (i.e the zero level set of ϕ) in such a form that it is constant normal to each level set of ϕ. This, which is easily realizable [26], is only done if the data is not already defined in the whole embedding space.

We will exemplify the framework with the simplest case, the heat flow or Laplace equation for scalar data defined on a surface. For scalar data u defined on the plane, that is, $u(x, y) : \Omega \subset I\!R^2 \rightarrow I\!R$, as we saw before, the heat flow is given by (5.1), and its corresponding energy by (5.2). If we now want to smooth scalar data u defined on a surface $\mathcal{S}$, that is, $u(x, y) : \mathcal{S} \rightarrow I\!R$, we must find the minimizer of the energy given by

$$\frac{1}{2} \int_{\mathcal{S}} \| \nabla_{\mathcal{S}} u \|^2 \, d\mathcal{S}. \tag{5.5}$$

The equation that minimizes this energy is its gradient descent flow (e.g., [176]):

$$\frac{\partial u}{\partial t} = \Delta_{\mathcal{S}} u. \tag{5.6}$$

Here $\nabla_{\mathcal{S}}$ is the intrinsic gradient and $\Delta_{\mathcal{S}}$ the intrinsic Laplacian or Laplace-Beltrami operator.

Classically, eq. (5.6) would be implemented in a triangulated surface, giving place to sophisticated and elaborated algorithms even for such simple flows. We now show how to simplify this when considering implicit representations.

Let $\vec{v}$ be a generic three dimensional vector, and $P_{\vec{v}}$ the operator that projects a given three dimensional vector onto the plane orthogonal to $\vec{v}$. It is then easy to show that the harmonic energy (5.5) ([40]) is equivalent to (see for example [174])

$$\frac{1}{2}\int_{\mathcal{S}} \parallel P_{\vec{N}}\nabla u \parallel^2 d\mathcal{S}, \tag{5.7}$$

where $\vec{N}$ is the normal to the surface $\mathcal{S}$. In other words, $\nabla_{\mathcal{S}} u = P_{\vec{N}}\nabla u$. That is, the gradient intrinsic to the surface ($\nabla_{\mathcal{S}}$) is just the projection onto the surface of the 3D Cartesian (classical) gradient ∇. We now embed this in the function ϕ:

$$\frac{1}{2}\int_{\mathcal{S}} \parallel \nabla_{\mathcal{S}} u \parallel^2 d\mathcal{S} = \frac{1}{2}\int_{\Omega\in I\!R^3} \parallel P_{\nabla\phi}\nabla u \parallel^2 \delta(\phi) \parallel \nabla\phi \parallel dx,$$

where $\delta(\cdot)$ stands for the delta of Dirac, and all the expressions above are considered in the sense of distributions. Note that first we got rid of intrinsic derivatives by replacing $\nabla_{\mathcal{S}}$ by $P_{\vec{N}}\nabla u$ (or $P_{\nabla\phi}\nabla u$) and then replaced the intrinsic integration ($\int_{\mathcal{S}} d\mathcal{S}$) by the explicit one ($\int_{\Omega\in I\!R^3} dx$) using the delta function. Intuitively, although the energy lives in the full space, the delta function forces the penalty to be effective only on the level set of interest. The gradient descent of this energy is given by

$$\frac{\partial u}{\partial t} = \frac{1}{\parallel \nabla\phi \parallel}\nabla\cdot(P_{\nabla\phi}\nabla u \parallel \nabla\phi \parallel). \tag{5.8}$$

In other words, this equation corresponds to the intrinsic heat flow for data on an implicit surface. But all the gradients in this PDE are defined in the three dimensional Cartesian space, not in the surface $\mathcal{S}$ (this is why we need the data to be defined at least on a band around the surface). The numerical implementation is then straightforward. Once again, due to the implicit representation, classic numerics are used, avoiding elaborate projections onto discrete surfaces and discretization on general meshes, e.g., [34, 81]. The same framework can be applied to other variational formulations as well as to PDE's defined on surfaces, e.g., the ones exemplified below [7].

A particularly interesting example is obtained when we have unit vectors defined on the surface. That is, we have data of the form $u : \mathcal{S} \to S^{n-1}$. When $n = 3$ our unit

vectors lie on the sphere. Following the work [178] for color images defined on the plane, we show in Figure 5.3 how to denoise a color image painted on an implicit surface. The basic idea is to normalize the RGB vector (a three dimensional vector) to a unit vector representing the chroma, and diffuse this unit vector with the harmonic maps flow embedded on the implicit surface extending the intrinsic heat flow example presented above. The corresponding magnitude, representing the brightness, is smoothed separately via scalar diffusion flows as those presented before for images on the plane (e.g., an intrinsic TV anisotropic heat flow). That is, we have to regularize a map from the zero level-set onto S^2 (the chroma) and another one onto $\mathbb{R}$ (the brightness).

Following the same framework and the work in [187, 188, 197], we show in Figure 5.4 the result of reaction diffusion flows solved on implicit surfaces in order to generate intrinsic patterns.

Finally, inspired by the work on line integral convolution [14] and that on anisotropic diffusion [154], the authors of [35] suggested to use anisotropic diffusion to visualize flows in 2D and 3D. The basic idea is, starting from a random image, anisotropically diffuse it in the directions dictated by the flow field. The authors presented very nice results both in 2D (flows on the plane) and 3D (flows on a surface), but once again using triangulated surfaces which introduce many computational difficulties. In a straightforward fashion we can compute these anisotropic diffusion equations on the implicit surfaces with the framework here introduced, and some results are presented in Figure 5.5.

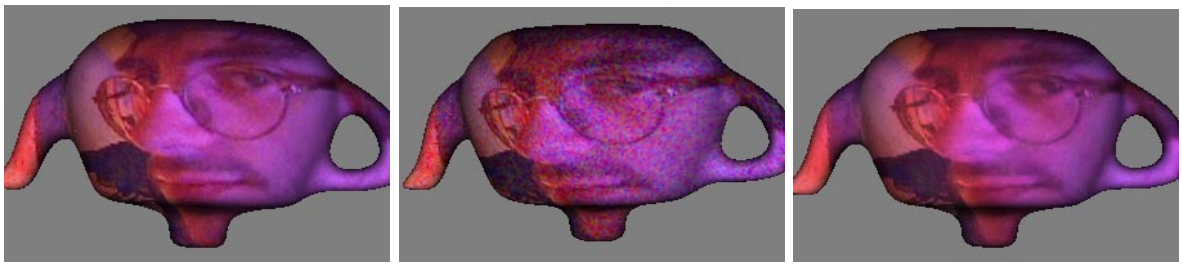


Figure 5.3: Intrinsic vector field regularization. Left: original color image. Middle: heavy noise has been added to the 3 color channels. Right: color image reconstructed after 20 steps of anisotropic diffusion of the chroma vectors.

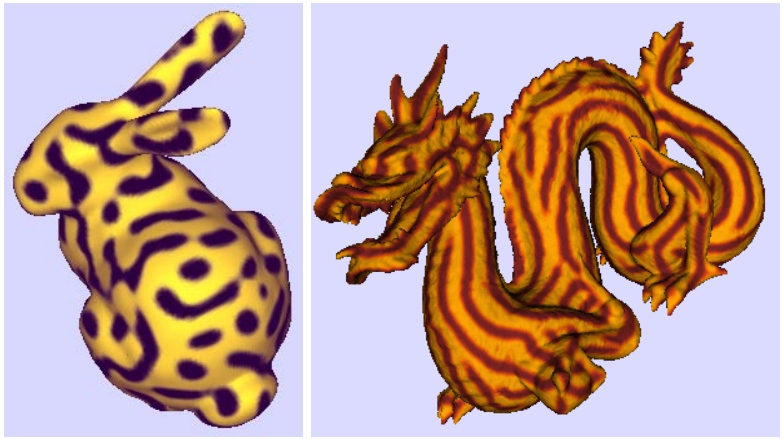


Figure 5.4: Texture synthesis via intrinsic reaction-diffusion flows on implicit surfaces. Left: isotropic. Right: anisotropic.

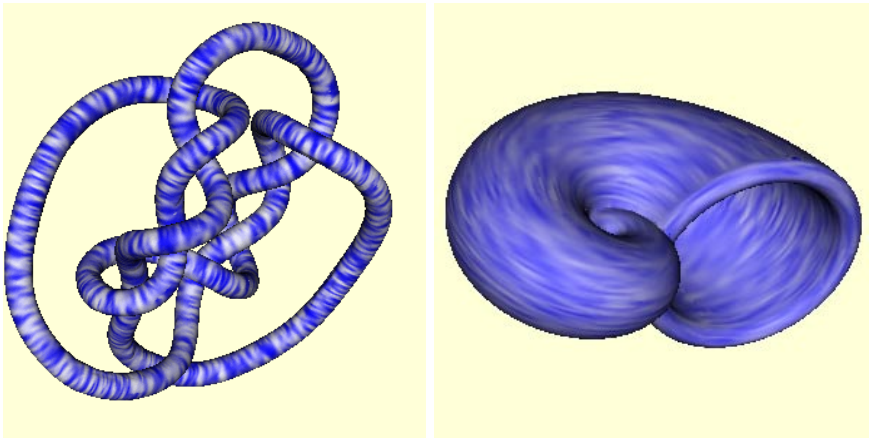


Figure 5.5: Flow visualization on implicit 3D surfaces via intrinsic anisotropic diffusion flows. Left: flow aligned with the major principal direction of the surface. Right: flow aligned with the minor principal direction of the surface. Pseudo-color representation of scalar data is used.

5.3 The Level-Set Method in Image Processing and Computer Vision

We now present a number of examples on the use of the level-set method described above for problems in image processing and computer vision.

5.3.1 Image Segmentation

One of the most popular applications of level-set methods in image processing and computer vision is for image segmentation. The contributions in this area started shortly after the work in [95] (which is one of the first papers in computer vision using the level-set method) by the works in [16, 115, 116]. These authors showed how to embed in the level-set framework the pioneering work on snakes and active contours by Terzopoulos and colleagues [91].

Consider the image on the left of Figure 5.6. Terzopoulos and colleagues suggested to detect the objects in this image (segment the image) starting with a curve that surrounds the object/s, and letting the curve deform (active-contour/snake) toward the boundary of the objects. The deformation is driven by the minimization of a given energy that penalizes non-smooth curves that do not sit at the objects boundaries. The authors of [91] proposed a Lagrangian implementation of the curve deformation process, while Caselles *et al.* and Malladi *et al.* pioneered the use of the level-set method for this approach. This added the classical topological freedom, thereby allowing the detection of multiple objects without prior knowledge of their number (later on Terzopoulos and colleagues showed a technique based on Lagrangian implementation to achieve this [118]). Following this work, in [17] (see also [18, 92, 93, 167, 180, 193] and [151] for pioneering extensions of this to object tracking), the authors showed that both approaches can be formally unified if one considers an energy given by

$$E(\mathcal{C}) = \int_{\mathcal{C}} g(\mathcal{C}) ds, \tag{5.9}$$

where ds is the Euclidean arc-length over the deforming curve $\mathcal{C} : [a, b] \to I\!R^2$ and $g(\cdot)$ is a function that penalizes curves that do not sit on the objects boundaries (a function of the image gradient for example). That is, image segmentation has been translated into finding a curve minimizing (5.9), thereby a geodesic in a space with metric $g(\cdot)$. The geodesic was computed using the level-set method. Examples are provided in Figure 5.6.

When describing image segmentation, variational problems, and PDE's, we can not avoid but think about the famous Mumford-Shah work [123], and ask ourself the relationship

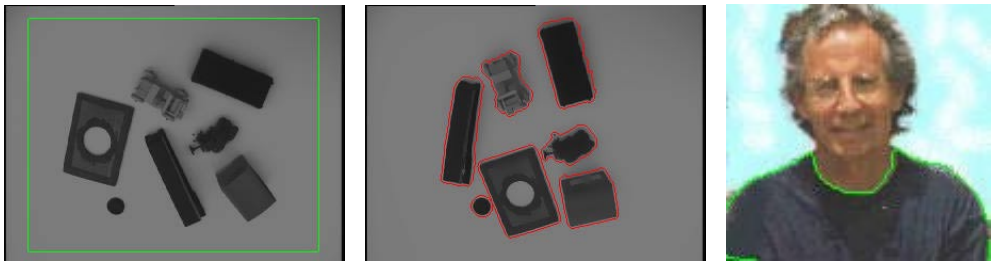


Figure 5.6: Level-set based object segmentation. The first figure on the left shows the original image and original contour, surrounding an un-known number of objects. The results of the geodesic active contours is given in the middle image. The image in the right is a result of the geodesic active contours framework implemented following Cohen and Kimmel.

between these techniques. Some of this relationship is described in [163], while additional one comes to light from recent works connecting the Mumford-Shah model and level-set techniques (see for example works by Paragios-Deriche, Yezzi *et al.*, and Chan-Vese). One of the works in this direction is presented in [23, 191]. This work is inspired in part by Zhao *et al.* [201]. In their work, multiple phases and their boundaries, represented via the level set method, evolve and interact in time, to minimize a bulk-surface energy. Combining several level set functions together, triple junctions were also represented and evolved in time. Inspired by this, Chan and Vese presented a multi-phase level set model for image segmentation. Triple junctions and complex topologies are segmented using more than one level set function. An example is provided in Figure 5.7. In this example, a multi-phase model with four phases is used, obtained by combining two level set functions. Here, the phases and their boundaries evolve in time, by minimizing an energy related to the Mumford and Shah piecewise-constant model for segmentation. We show the evolution of the curves and of the four phases, in a level set framework.

5.3.2 Stereo and 3D Reconstruction from Multiple Views

The problem of stereo is as follows: Recover the geometry of the scene when two or more images from the world are taken simultaneously. Since the internal parameters of the camera are unknown, the problem is essentially one of establishing correspondence between the

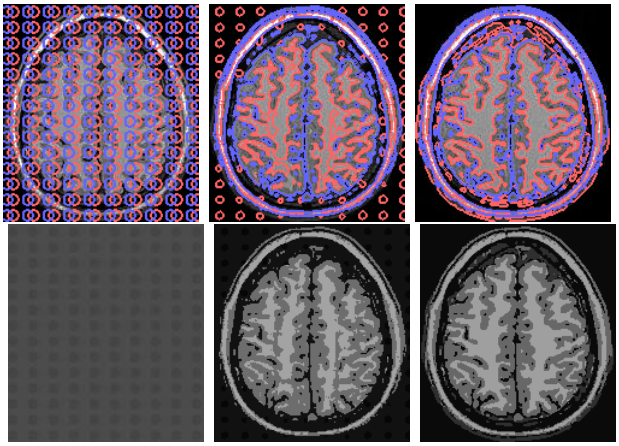


Figure 5.7: Evolution of the four-phase segmentation model from [23], using two level set functions: evolving curves (top) and phases (bottom).

views. The correspondence problem is usually addressed setting up a functional and looking for its extrema. Once the correspondence has been achieved, the 3D point is reconstructed by intersecting the corresponding optical rays. See for example [52, 68, 79] for further geometric details on the old problem of stereo.

Faugeras and Keriven pioneered the use of level-sets and the geodesic framework for this problem. They proposed to start from some initial 3D surface $\mathcal{S}_0$ and deform it toward the minimization of a given geometric functional. One of the functionals proposed in [54] reads as follows:

$$E(\mathcal{S},\vec{\mathcal{N}}) = \int\int \Psi(\mathcal{S},\vec{\mathcal{N}})da = -\sum_{i,j=1,i\neq j}^{n} \frac{\langle I_i, I_j\rangle}{\langle I_i, I_i\rangle\langle I_j, I_j\rangle}, \tag{5.10}$$

where i, j run over all the n available images I_i, and $\langle\cdot,\cdot\rangle$ is the cross correlation between the images, which includes the geometry of the perspective projection and the assumption of Lambertian surfaces [54]. Note that this approach consists basically on replacing the edge dependent 'metric' g we used for the segmentation approach by a new 'metric' Ψ that favors correlation between the collection of images. The authors work out the Euler-Lagrange equations for this formulation and embed it in the level-set framework. The example in Figure 5.8 was provided by Ronny Kimmel, and it corresponds to a simplification of the model by Faugeras-Keriven that he has recently proposed (additional examples can be found

in the home pages of Faugeras and Keriven). This work was also extended by [198], where the authors explicitly combine multi-view reconstruction with segmentation ideas as those described before. The authors suggest to find a surface whose projection properly segments the multiple given images, and an example from their work is given in Figure 5.9.

Figure 5.8: 3D reconstruction from a stereo pair using the geodesic stereo approach. First, the stereo pair is shown, followed by the reconstructed 3D shape.

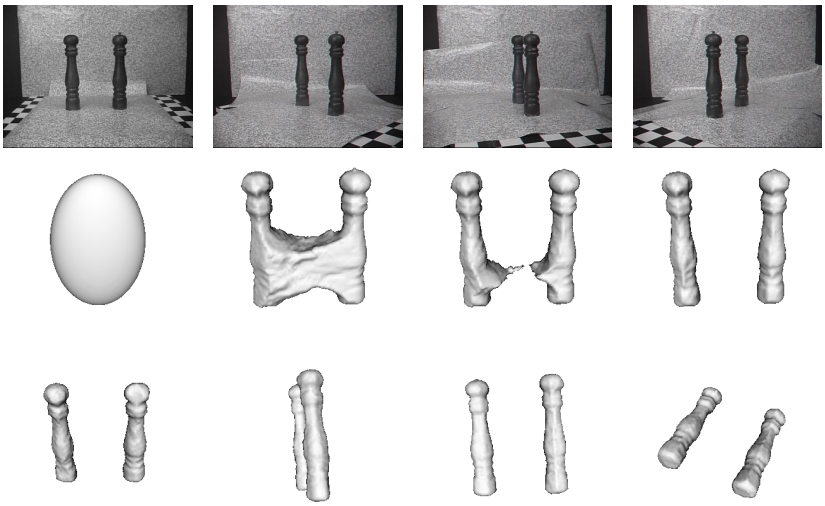


Figure 5.9: Surface reconstruction from multiple-views. The first row shows four different views. The second row shows four steps of the evolution, while the last row shows four different 3D views of the reconstructed surface.

5.4 Shape from Shading

According to the so called *Lambertian* shading rule, the 2D array of pixel gray levels, corresponding to the shading of a 3D object, is proportional to the cosine of the angle between the light source direction and the surface normal. The shape from shading problem is the inverse problem of reconstructing the 3D surface from this shading data. The history of this problem is extensive. We here described a basic technique, developed by Kimmel and Bruckstein [96] to address this problem. An outstanding contribution to the problem was done in [158], based on the theory of viscosity solutions (this is currently being extended at INRIA by Faugeras and his collaborators; see also [125].) See these references for details and an extensive literature.

Consider a smooth surface, actually a graph, given by $z(x, y)$. According to the Lambertian shading rule, the shading image $I(x, y)$ is equal (or proportional) to the inner product between the light direction $\hat{l} = (0, 0, 1)$ and the normal $\vec{\mathcal{N}}(x, y)$ to the parameterized surface. This gives the so called *irradiance equation*:

$$I(x, y) = \hat{l} \cdot \vec{\mathcal{N}} = \frac{1}{\sqrt{1 + p^2 + q^2}},$$

where $p := \partial z/\partial x$ and $q := \partial z/\partial y$. Starting from a small circle around a singular point, Bruckstein [12] observed that equal height contours $\mathcal{C}(p, t) : S \to I\!R^2$ of the surface z (t stands for the height) hold

$$\frac{\partial \mathcal{C}}{\partial t} = \frac{I}{\sqrt{1 - I^2}} \vec{n},$$

where now $\vec{n}$ is the 2D unit normal to the equal height contour (or level-set of z). This means that the classical shape from shading problem is simply a curve evolution problem, and as so, we can use all the curve evolution machinery to solve it. In particular, we can use both the level-set and the fast marching numerical techniques (the weight for the distance is always positive and given by $\sqrt{1/I^2 - 1}$). An example, courtesy of the authors of [96], is presented in Figure 5.10.

We conclude by mentioning that this work by Kimmel and Bruckstein on shape from shading using curve evolution and level-sets inspired in part the work in [137]. This presents the general connection between the unsteady and steady approaches to curve and surface evolution.

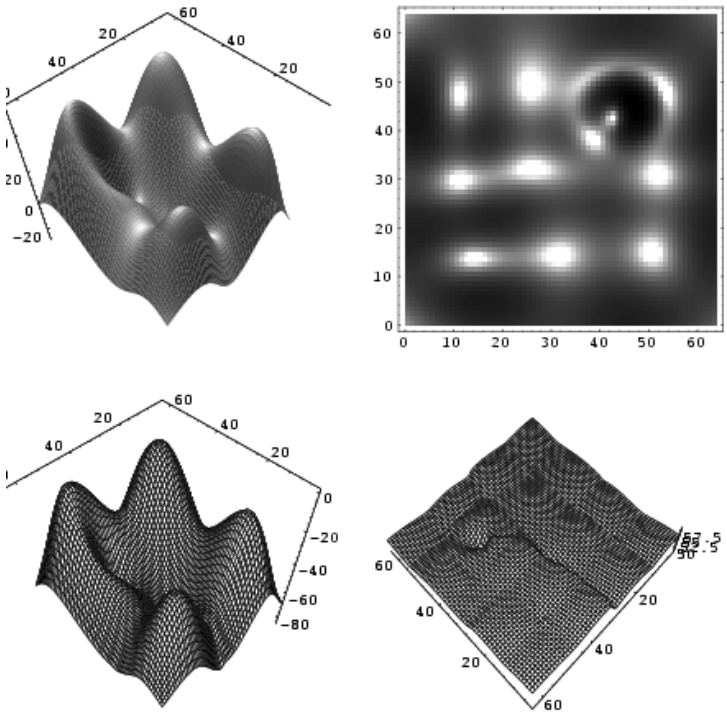


Figure 5.10: Example of shape from shading via curve evolution. The figure shows the original surface, the simulated shading, the reconstructed surface, and the reconstruction error.

References

[1] Adalsteinsson, D. and Sethian, J., *A fast level set method for propagating interfaces*, J. Comput. Phys. 118, 269-277 (1995).

[2] Alvarez, L., Guichard, F., Lions, P.L. and Morel, J.M., *Axioms and fundamental equations of image processing*, Arch. Rational Mechanics 123, 199-257 (1993).

[3] Alvarez, L., Lions, P.L. and Morel, J.M., *Image selective smoothing and edge detection by nonlinear diffusion*, SIAM J. Numer. Anal. 29, 845-866 (1992).

[4] Barles, G. and Georgelin, C., *A simple proof of convergence for an approximation scheme for computing motions by mean curvature*, SIAM J. Numer. Anal. 32, 484-500 (1995).

[5] Balsara, D. and Shu, C.-W., *Monotonicity preserving weighted essentially non-oscillatory schemes with increasingly high order of accuracy*, J. Comput. Phys. 160, 405-452 (2000).

[6] Bardi, M. and Osher, S., *The nonconvex multi-dimensional Riemann problem for Hamilton-Jacobi equations*, SIAM J. Numer. Anal. 22, 344-351 (1991).

[7] Bertalmio, M, Cheng, L.T., Osher, S. and Sapiro, G., *Variational problems and partial differential equations on implicit surfaces*, J. Comput. Phys., to appear.

[8] Black, M., Sapiro, G., Marimont, D. and Heeger, D., *Robust anisotropic diffusion*, IEEE Trans. Image Processing 7:3, 421-432 (1998).

[9] Bloomenthal, J., *Introduction to Implicit Surfaces*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997.

[10] Boris, J.P. and Book, D.L., *Flux corrected transport I, SHASTA, a fluid transport algorithm that works*, J. Comput. Phys. 11, 38-69 (1973).

[11] Brenier, Y. and Osher, S., *The discrete one-sided Lipschitz condition for convex scalar conservation laws*, SIAM J. Numer. Anal. 25, 8-23 (1988).

[12] Bruckstein, A.M., *On shape from shading*, Comp. Vision Graph. Image Processing 44, 139-154 (1988).

[13] Burchard, P., Cheng, L.-T., Merriman, B. and Osher, S., *Motion of curves in three spatial dimensions using a level set approach*, J. Comput. Phys. 165, 463-502 (2001).

[14] Cabral, B. and Leedom, C., *Imaging vector fields using line integral convolution*, ACM Computer Graphics (SIGGRAPH '93) 27:4, 263-272 (1993).

[15] Caiden, R., Fedkiw, R. and Anderson, C., *A numerical method for two-phase flow consisting of separate compressible and incompressible regions*, J. Comput. Phys. 166, 1-27 (2001).

[16] Caselles, V., Catte, F., Coll, T. and Dibos, F., *A geometric model for active contours*, Numerische Mathematik 66, 1-31 (1993).

[17] Caselles, V., Kimmel, R. and Sapiro, G., *Geodesic active contours*, International Journal of Computer Vision 22:1, 61-79 (1997).

[18] Caselles, V., Kimmel, R., Sapiro, G. and Sbert, C., *Minimal surfaces based object segmentation*, IEEE-PAMI 19:4, 394-398 (1997).

[19] Caselles, V., Morel, J.M., Sapiro, G. and Tannenbaum, A., Editors, *Special Issue on Partial Differential Equations and Geometry-Driven Diffusion in Image Processing and Analysis*, IEEE Trans. Image Processing 7, March 1998.

[20] Cercignani, C., Gamba, I., Jerome, J. and Shu, C.-W., *Device benchmark comparisons via kinetic, hydrodynamic, and high-field models*, Comput. Meth. Appl. Mech. Engin. 181, 381-392 (2000).

[21] Chakravarthy, S. and Osher, S., *Numerical experiments with the Osher upwind scheme for the Euler equations*, AIAA J. 21, 1241-1248 (1983).

[22] Chan, T. and Vese, L., *Active contour model without edges*, in Lect. Notes in C.S., edited by M. Neilsen, P. Johansen, O.F. Olson and J. Weickert, Springer-Verlag, Berlin/New York, v. 1687, p. 141 (1999).

[23] Chan, T. and Vese, L., *Image segmentation using level sets and the piecewise-constant Mumford-Shah model*, UCLA CAM Report 00-14, 2000.

[24] Chang, Y., Hou, T., Merriman, B. and Osher, S., *A level set formulation of Eulerian interface capturing methods for incompressible fluid flows*, J. Comput. Phys. 124, 449-464 (1996).

[25] Chen, S., Merriman, B., Kang, M., Caflisch, R., Ratsch, C., Guyre, M. Fedkiw, R., Anderson, C. and Osher, S., *A level set method for thin film epitaxial growth*, J. Comput. Phys. 167, 475-500 (2001).

[26] Chen, S., Merriman, B., Osher, S. and Smereka, P., *A simple level set method for solving Stefan problems*, J. Comput. Phys. 135, 8-29 (1997).

[27] Chen, Y. G., Giga, Y. and Goto, S., *Uniqueness and existence of viscosity solutions of generalized mean curvature flow equations*, J. Diff. Geom. 33, 749-786 (1991).

[28] Cheng, L.-T., Burchard, P., Merriman, B. and Osher, S., *Motion of curves constrained on surfaces using a level set approach*, UCLA CAM Report 00-32, J. Comput. Phys., to appear 2001.

[29] Chopp, D., *Computing minimal surfaces via level set curvature flow*, J. Comput. Phys. 106, 77-91 (1993).

[30] Crandall, M. and Lions, P.-L., *Viscosity solutions of Hamilton-Jacobi equations*, Trans. Amer. Math. Soc. 277, 1-42 (1983).

[31] Crandall, M. and Lions, P.-L., *Two approximations of solutions of Hamilton-Jacobi equations*, Math. Comp. 43, 1-19 (1984).

[32] Crandall, M. and Majda, A., *Monotone difference approximations for scalar conservation laws*, Math. Comput. 34, 1-21 (1980).

[33] Deacon, A.G. and Osher, S., *A finite element method for a boundary value problem of mixed type*, SIAM J. Numer. Anal. 16, 756-778 (1979).

[34] Desbrun, M., Meyer, M., Schröder, P., and Barr, A.H., *Discrete differential-geometry operators in nD*, Multi-res modeling group TR, Caltech, September 2000 (obtained from www.multires.caltech.edu).

[35] Diewald, U., Preufer, T. and Rumpf, M., *Anisotropic diffusion in vector field visualization on Euclidean domains and surfaces*, IEEE Trans. Visualization and Computer Graphics 6, 139-149 (2000).

[36] Dorsey J. and Hanrahan, P., *Digital materials and virtual weathering*, Scientific American 282:2, 46-53 (2000).

[37] Durlofsky, L.J., Engquist, B. and Osher, S., *Triangle based adaptive stencils for the solution of hyperbolic conservation laws*, J. Comput. Phys. 98, 64-73 (1992).

[38] Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M. and Stuetzle, W., *Multi-resolution analysis of arbitrary meshes*, Computer Graphics (SIGGRAPH '95 Proceedings), 173-182, 1995.

[39] Eck, M. and Hoppe, H., *Automatic reconstruction of B-spline surfaces of arbitrary topological type*, Computer Graphics, 1996.

[40] Eells, J. and Lemarie, L., *A report on harmonic maps*, Bull. London Math. Soc. 10:1, 1-68 (1978).

[41] Engquist, B., Fatemi, E. and Osher, S., *Numerical solution of the high frequency asymptotic expansion for hyperbolic equations*, Proc. 10th Ann. Review of Progress in Applied Comp. Electromagnetics, Monterey, CA, 32-45 (1994).

[42] Engquist, B. and Osher, S., *Stable and entropy satisfying approximations for transonic flow calculations*, Math. Comp. 34, 45-57 (1980).

[43] Engquist, B. and Osher, S., *One-sided difference approximations for nonlinear conservation laws*, Math. Comp. 36, 321-351 (1981).

[44] Engquist, B., Osher, S. and Zhong, S., *Fast wavelet based algorithms for linear evolution equations*, SIAM J. Sci. Stat. Comput. 15:4, 755-775 (1994).

[45] Engquist, B., Runborg, O. and Tornberg, A.-K., *High frequency wave propagation by the segment projection method*, UCLA CAM Report 01-13, (2001), submitted to J. Comput. Phys.

[46] Enright, D., Fedkiw, R., Ferziger, J. and Mitchell, I. *A hybrid particle level set method for improved interface capturing*, J. Comput. Phys., in review.

[47] Evans, L.C., *Convergence of an algorithm for mean curvature motion*, Indiana University Mathematics Journal 42, 553-557 (1993).

[48] Evans, L.C. and Spruck, J., *Motion of level sets by mean curvature, I*, J. Differential Geometry 33, 635-681 (1991).

[49] Fatemi, E., Engquist, B. and Osher, S., *Numerical solution of the high frequency asymptotic expansion of the scalar wave equation*, J. Comput. Phys. 120, 145-155 (1995).

[50] Fatemi, E., Engquist, B. and Osher, S., *Finite difference methods for the nonlinear equations of perturbed geometric optics*, ACES J. 11, 90-98 (1996).

[51] Fatemi, E., Jerome, J. and Osher, S., *Solution of the hydrodynamic device model using high order nonoscillatory shock capturing algorithms*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 10, 232-243 (1991).

[52] Faugeras, O.D. *Three-Dimensional Computer Vision: A Geometric Viewpoint*, MIT Press, Cambridge, MA, 1993.

[53] Faugeras, O., Clément, F., Deriche, R., Keriven, R., Papadopoulo, T., Gomes, J., Hermosillo, G., Kornprobst, P. Lingrad, D., Roberts, J. Viéville, T. and Devernay, F., *The inverse EEG and MEG problems: The adjoint state approach I: The continuous case*, INRIA Research Report 3673, June 1999.

[54] Faugeras, O.D. and Keriven, R., *Variational principles, surface evolution, PDE's, level-set methods, and the stereo problem*, IEEE Trans. Image Processing 7:3, 336-344 (1998).

[55] Fedkiw, R., *Coupling an Eulerian fluid calculation to a Lagrangian solid calculation with the ghost fluid method*, J. Comput. Phys., in review.

[56] Fedkiw, R., Aslam, T., Merriman, B. and Osher, S., *A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method)*, J. Comput. Phys. 152, 457-492 (1999).

[57] Fedkiw, R., Aslam, T., and Xu, S., *The ghost fluid method for deflagration and detonation discontinuities*, J. Comput. Phys. 154, 393-427 (1999).

[58] Fedkiw, R., Merriman, B. and Osher, S., *High accuracy numerical methods for thermally perfect gas flows with chemistry*, J. Comput. Phys. 132, 175-190 (1997).

[59] Fedkiw, R., Merriman, B. and Osher, S., *Efficient characteristic projection in upwind difference schemes for hyperbolic systems; the complementary projection method*, J. Comput. Phys. 141, 22-36 (1998).

[60] Foster, N. and Fedkiw, R., *Practical animation of liquids*, Siggraph 2001 Annual Conference (2001).

[61] Frisken, S.F., Perry, R.N., Rockwood, A. and Jones, T., *Adaptively sampled fields: A general representation of shape for computer graphics*, Computer Graphics (SIGGRAPH), New Orleans, July 2000.

[62] Gabor, D., *Information theory in electron microscopy*, Laboratory Investigation 14, 801-807 (1965).

[63] Giga, Y. and Sato, M.-H., *A level set approach to semicontinuous solutions for Cauchy problems*, to appear in Comm. in PDE (2001).

[64] Godunov, S.K. and Ryabenkii, V.S., *Spectral criteria for the stability of boundary problems for non self-adjoint difference equations*, Uspekhi Mat. Nauk 18, (1963).

[65] Gottlieb, S., Shu, C-W. and Tadmor, E., *Strong stability preserving high order time discretization methods*, SIAM Review 43, 89-112 (2001).

[66] Grayson, M., *The heat equation shrinks embedded plane curves to round points*, J. Diff. Geom. 26, 285-314 (1987).

[67] Green, M. and Osher, S., *Steiner polynomials, Wulff flows and some new isoperimetric inequalities for convex plane curves*, Asian J. Math. 3, 659-676 (1999).

[68] Grimson, W.E.L., *From Images to Surfaces*, MIT Press, Cambridge, MA, 1981.

[69] Guichard. F. and Morel, J.M., *Introduction to Partial Differential Equations in Image Processing*, Tutorial Notes, *IEEE Int. Conf. Image Proc.*, Washington, DC, October 1995.

[70] Gustafsson, B., Kreiss, H.-O. and Sundstrom, A., *Stability theory of difference approximations for mixed initial boundary value problems. II*, Math. Comp. 26, 649-686 (1972).

[71] Gyure, M., Ratsch, C., Merriman, B., Caflisch, R., Osher, S., Zinck, J. and Vvedensky, D., *Level set Methods for the simulation of epitaxial phenomena*, Phys. Rev. E. 58, 6927-6930 1998.

[72] Harabetian, E. and Osher, S., *Regularization of ill-posed problems via the level set approach*, SIAM J. Appl. Math. 58, 1689-1706 (1998).

[73] Harabetian, E., Osher, S. and Shu, C.-W., *An Eulerian approach for vortex motion using a level set regularization procedure*, J. Comput. Phys. 127, 15-26 (1996).

[74] Harten, A., *High resolution schemes for hyperbolic conservation laws*, J. Comput. Phys. 49, 357-393 (1983).

[75] Harten, A. and Osher, S., *Uniformly high-order accurate non-oscillatory schemes, I*, SIAM J. Numer. Anal. 24, 279-304 (1987).

[76] Harten, A., Engquist, B., Osher, S. and Chakravarthy, S., *Uniformly high-order accurate essentially non-oscillatory schemes III*, J. Comput. Phys. 71, 231-303 (1987).

[77] Harten, A., Osher, S., Engquist, B. and Chakravarthy, S., *Some results on uniformly high order accurate essentially non-oscillatory schemes*, Appl. Numer. Math. 2, 347-377 (1986).

[78] Helmsen, J., Puckett, E., Colella, P. and Dorr, M., *Two new methods for simulating photolithography development in 3D*, Proc. SPIE 2726, 253-261 (1996).

[79] Horn, B.K.P., *Robot Vision*, MIT Press, Cambridge, MA, 1986.

[80] Hu, C. and Shu, C.-W., *Weighted essentially non-oscillatory schemes on triangular meshes*, J. Comput. Phys. 150, 97-127 (1999).

[81] Huiskamp, G., *Difference formulas for the surface Laplacian on a triangulated surface*, J. Comput. Phy. 95, 477-496 (1991).

[82] Hummel, R.A., *Representations based on zero-crossings in scale-space*, Proc. IEEE CVPR, 204-209 (1986).

[83] Ishii, H., *A generalization of Bence, Merriman, and Osher algorithm for motion by mean curvature*, In A. Damlamian, J. Spruck, and A. Visintin, Editors, *Curvature Flows and Related Topics*, pp. 111-127, Gakkôtosho, Tokyo, 1995.

[84] H. Ishii, G. E. Pires, and P. E. Souganidis, *Threshold dynamics type schemes for propagating fronts*, J. Math. Soc. Japan 51(2), 267-308 (1999).

[85] Jain, A.K., *Partial differential equations and finite-difference methods in image processing, part 1: Image representation*, J. of Optimization Theory and Applications 23, 65-91 (1977).

[86] Jerome, J. and Shu, C.-W., *Transport effects and characteristic modes in the modeling and simulation of submicron devices*, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 14, 917-923 (1995).

[87] Jiang, G.-S. and Peng, D., *Weighted ENO schemes for Hamilton Jacobi equations*, SIAM J. Sci. Comput. 21, 2126-2143 (2000).

[88] Jiang, G.-S. and Shu, C.-W., *Efficient implementation of weighted ENO schemes*, J. Comput. Phys. 126, 202-228 (1996).

[89] Kang, M., Fedkiw, R., and Liu, X.-D., *A boundary condition capturing method for multiphase incompressible flow*, J. Sci. Comput. 15, 323-360 (2000).

[90] Karni, S., *Multicomponent flow calculations by a consistent primitive algorithm*, J. Comput. Phys. 112, 31-43 (1994).

[91] Kass, M., Witkin, A. and Terzopoulos, D., *Snakes: Active contour models*, International Journal of Computer Vision 1, 321-331 (1988).

[92] Kichenassamy, S., Kumar, A., Olver, P., Tannenbaum, A. and Yezzi, A., *Gradient flows and geometric active contour models*, Proc. Int. Conf. Comp. Vision '95, 810-815, Cambridge, June 1995.

[93] Kichenassamy, S., Kumar, A., Olver, P., Tannenbaum, A. and Yezzi, A., *Conformal curvature flows: from phase transitions to active vision*, Archive for Rational Mechanics and Analysis 134, 275-301 (1996).

[94] Kim, Y.-T., Goldenfeld, N. and Dantzig, J., *Computation of dendritic microstructures using a level set method*, Phys. Rev. E 62, 2471-2474 (2000).

[95] Kimia, B. B., Tannenbaum, A. and Zucker, S. W. *Toward a computational theory of shape: An overview*, Lecture Notes in Computer Science 427, 402-407, Springer-Verlag, New York, 1990.

[96] Kimmel, R. and Bruckstein, A.M., *Tracking level sets by level sets: A method for solving the shape from shading problem*, CVIU 62(1), 47-58 (1995).

[97] Kimmel, R., *Numerical geometry of images: Theory, algorithms, and applications*, Technion CIS Report 9910, October 1999.

[98] Koenderink, J.J. *The structure of images*, Biological Cybernetics 50, 363-370 (1984).

[99] Kreiss, H.-O., *Difference approximations for the initial-boundary value problem for hyperbolic differential equations*, in Numerical Solutions of Nonlinear Differential Equations. Ed., D. Greenspan, John Wiley, New York, 141-166 (1966).

[100] Krishnamurthy, V. and Levoy, M., *Fitting smooth surfaces to dense polygon meshes*, Proc SIGGRAPH '96 (New Orleans, LA), in Computer Graphics Proceedings, ACM SIGGRAPH, 313-324 (1996).

[101] Kupka, I.A.K. and Osher, S., *On the wave equation in a multi-dimensional corner*, Comm. Pure Appl. Math. 24, 381-393 (1971).

[102] Lafon, F. and Osher, S., *High order two dimensional nonoscillatory methods for solving Hamilton-Jacobi equations*, J. Comput. Phys. 123, 235-253 (1996).

[103] Lagnado, R. and Osher, S., *Reconciling differences*, Risk Magazine 10, 79-83 (1997).

[104] Lagnado, R. and Osher, S., *A technique for calibrating derivative security pricing models, numerical solution of an inverse problem*, J. Comput. Finance 1, 13-25 (1997).

[105] Lindeberg, T., *Scale-Space Theory in Computer Vision*, Kluwer, 1994.

[106] Liu, X.-D. and Osher, S., *Nonoscillatory high order accurate self similar maximum principle satisfying shock capturing schemes*, SIAM J. Numer. Anal. 33, 760-779 (1996).

[107] Liu, X.-D. and Osher, S., *Convex ENO high-order multidimensional schemes without field by field projection or staggered grids*, J. Comput. Phys. 142, 304-330 (1998).

[108] Liu, X.-D., Osher, S. and Chan, T. *Weighted essentially non-oscillatory schemes*, J. Comput. Phys. 115, 200-212 (1994).

[109] Liu, X.-D., Fedkiw, R. and Kang, M., *A boundary condition capturing method for Poisson's equation on irregular domains*, J. Comput. Phys. 160, 151-178 (2000).

[110] Majda, A., McDonough, J. and Osher, S., *The Fourier method for nonsmooth initial data*, Math. Comp. 32, 1041-1081 (1978).

[111] Majda, A. and Osher, S., *Reflections of singularities at the boundary*, Comm. Pure Appl. Math. 28, 479-499 (1975).

[112] Majda, A. and Osher, S., *Initial-boundary value problems for hyperbolic equations with uniformly characteristic boundary*, Comm. Pure Appl. Math. 28, 607-675 (1975).

[113] Majda, A. and Osher, S., *Propagation of error into regions of smoothness for accurate difference approximations to hyperbolic equations*, Comm. Pure Appl. Math. 30, 671-705 (1977).

[114] Majda, A. and Osher, S., *A systematic approach for correcting nonlinear instabilities: the Lax-Wendroff scheme for scalar conservation laws*, Numer. Math. 30, 429-452 (1978).

[115] Malladi, R., Sethian, J.A. and Vemuri, B.C., *Evolutionary fronts for topology independent shape modeling and recovery*, Proc. of the 3rd ECCV, Stockholm, Sweden, pp.3-13, 1994.

[116] Malladi, R., Sethian, J.A. and Vemuri, B.C., *Shape modeling with front propagation: A level set approach*, IEEE Trans. on PAMI 17, 158-175 (1995).

[117] Mauch, S., *Closest point transform*, www.ama.caltech.edu/∼seanm/software/cpt/cpt.html.

[118] McInerney, T. and Terzopoulos, D., *Topologically adaptable snakes*, Proc. ICCV, Cambridge, MA, June 1995.

[119] Merriman, B., Bence, J. and Osher, S., *Diffusion generated motion by mean curvature*, in J. E. Taylor, Editor, *Computational Crystal Growers Workshop*, pp. 73-83, American Mathematical Society, Providence, Rhode Island, 1992.

[120] Merriman, B., Bence, J. and Osher, S., *Motion of multiple junctions: A level-set approach*, J. Comput. Phys. 112, 334-363 (1994).

[121] Merriman, B., Caflisch, R. and Osher, S. *Level set methods with an application to modeling the growth of thin films*, in "Free boundary value problems, theory and applications", eds. I. Athanasopoulos, G. Makreakis and J. Rodrigues (CRC Press, Boca Raton), pp. 51-70 (1999).

[122] Mulder, W., Osher, S., and Sethian, J., *Computing interface motion in compressible gas dynamics*, J. Comput. Phys. 100, 209-228 (1992).

[123] Mumford, D. and Shah, J., *Optimal approximations by piecewise smooth functions and variational problems*, Comm. Pure Appl. Math. 42, 577-685 (1989).

[124] Nguyen, D., Fedkiw, R. and Kang, M., *A boundary condition capturing method for incompressible flame discontinuities*, J. Comput. Phys. 172, 71-98 (2001).

[125] Oliensis, J. and Dupuis, P., *Direct method for reconstructing shape from shading*, Proceedings SPIE Conf. 1570 on Geometric Methods in Computer Vision, 116-128, 1991.

[126] Osher, S., *Systems of difference equations with general homogeneous boundary conditions*, Trans. Amer. Math. Soc. 137, 177-201 (1969).

[127] Osher, S., *On systems of difference equations with wrong boundary conditions*, Math. Comp. 23, 335-340 (1969).

[128] Osher, S., *Stability of parabolic difference approximations to certain mixed initial boundary value problems*, Math. Comp. 26, 13-39 (1972).

[129] Osher, S., *Initial boundary value problems for hyperbolic systems in regions with corners, I.* Trans. Amer. Math. Soc. 176, 141-164 (1973).

[130] Osher, S., *On Green's function for the biharmonic equation in a right angle wedge*, J. Math. Anal. Appl. 43, 705-716 (1973).

[131] Osher, S., *Initial boundary value problems for hyperbolic systems in regions with corners, II.* Trans. Amer. Math. Soc. 198, 151-175 (1974).

[132] Osher, S., *Boundary value problems for equations of mixed type: I. The Lavrentev-Bitsadze model*, Comm. in P.D.E. 2, 499-547 (1977).

[133] Osher, S., *Numerical solution of singular perturbation problems and one-sided difference schemes*, North Holland Math. Studies 47, 1981.

[134] Osher, S., *The Riemann problem for nonconvex scalar conservation laws and the Hamilton-Jacobi equations*, Proc. Amer. Math. Soc. 89, 641-646 (1983).

[135] Osher, S., *Riemann solvers, the entropy condition and difference approximations*, SIAM J. Numer. Anal. 21, 217-235 (1984).

[136] Osher, S., *Convergence of generalized MUSCL schemes*, SIAM J. Numer. Anal. 22, 947-961 (1985).

[137] Osher, S., *A level set formulation for the solution of the Dirichlet problem for Hamilton-Jacobi equations*, SIAM J. Anal. 24, 1145-1152 (1993).

[138] Osher, S. and Chakravarthy, S., *Upwind schemes and boundary conditions with applications to Euler equations in general geometries*, J. Comput. Phys. 50, 447-481 (1983).

[139] Osher, S. and Chakravarthy, S., *High resolution schemes and the entropy condition*, SIAM J. Numer. Anal. 21, 955-984 (1984).

[140] Osher, S. and Chakravarthy, S., *Very high order accurate TVD schemes*, in IMA Volumes in Mathematics and Its Applications, 2, 229-274 (1986), Springer-Verlag.

[141] Osher, S., Cheng, L.-T., Kang, M., Shim, H. and Tsai, Y.-H., *Geometric optics in a phase space and Eulerian framework*, report of LS, Inc #LS-01-01, (Sept. 2001), submitted to J. Comput. Phys.

[142] Osher, S., Hafez, M. and Whitlow Jr., W., *Entropy condition satisfying approximations for the full potential equation of transonic flow*, Math. Comp. 44, 1-29 (1985).

[143] Osher, S. and Merriman, B., *The Wulff shape as the asymptotic limit of a growing crystalline interface*, Asian J. Math. 1, 560-571 (1997).

[144] Osher, S. and Rudin, R.T., *Feature-oriented image enhancement using shock filters*, SIAM J. Numer. Anal. 27, 919-940 (1990).

[145] Osher, S. and Sanders, R., *Numerical approximations to nonlinear conservation laws with locally varying time and space grids*, Math. Comp. 41, 321-336 (1983).

[146] Osher, S. and Santosa, F., *Level set method for optimization problems involving geometry and constraints, I. Frequencies of a two-density inhomogeneous drum*, J. Comput. Phys. 171, 277-288 (2001).

[147] Osher, S. and Sethian, J., *Fronts propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulations*, J. Comput. Phys. 79, 12-49 (1988).

[148] Osher, S. and Shu, C.-W., *High order essentially non-oscillatory schemes for Hamilton-Jacobi equations*, SIAM J. Numer. Anal. 28, 902-921 (1991).

[149] Osher, S. and Solomon, F., *Upwind difference schemes for hyperbolic systems of conservation laws*, Math. Comp. 38, 339-374 (1982).

[150] Osher, S. and Tadmor, E., *On the convergence of difference approximations to scalar conservation laws*, Math. Comp. 50, 19-51 (1988).

[151] Paragios, N. and Deriche, R., *A PDE-based level-set approach for detection and tracking of moving objects*, Proc. Int. Conf. Comp. Vision '98, Bombay, India, January 1998.

[152] Peng, D., Osher, S., Merriman, B, and Zhao, H.-K., *The geometry of Wulff crystal shapes and its relations with Riemann problems*, in Contemporary Math., edited by G.Q. Chen and E. DeBenedetto, AM. Math. Soc. Providence, R.I., v. 238, p. 251 (1999).

[153] Peng, D., Merriman, B., Osher, S., Zhao, H.-K., and Kang, M., *A PDE-based fast local level set method*, J. Comput. Phys. 155, 410-438 (1999).

[154] Perona, P. and Malik, J., *Scale-space and edge detection using anisotropic diffusion*, IEEE Trans. Pattern. Anal. Machine Intell. 12, 629-639 (1990).

[155] Praun, E., Finkelstein, A. and Hoppe, H., *Lapped textures*, ACM Computer Graphics (SIGGRAPH), New Orleans, July 2000.

[156] Price, C.B., Wambacq, P. and Oosterlink, A. *Image enhancement and analysis with reaction-diffusion paradigm*, IEE Proc. 137, 136-145 (1990).

[157] Romeny, B., Editor, *Geometry Driven Diffusion in Computer Vision*, Kluwer, 1994.

[158] Rouy, E. and Tourin, A., *A viscosity solutions approach to shape-from-shading*, SIAM J. Num. Anal. 29, 867-884 (1992).

[159] Rudin, L.I., Osher, S. and Fatemi, E., *Nonlinear total variation based noise removal algorithms*, Phyica D 60, 259-268 (1992).

[160] Ruuth, S., Fedkiw, R., Xin, J. and Osher, S., *A diffusion generated approach to the curvature motion of filaments*, J. Nonlinear Science, to appear.

[161] Ruuth, S., Merriman, B. and Osher, S., *Convolution generated motion as a link between cellular automata and continuum pattern dynamics*, J. Comput. Phys. 151, 836-861 (1999).

[162] Ruuth, S., Merriman, B. and Osher, S., *A fixed grid method for capturing the motion of self-intersecting interfaces and related PDEs*, J. Comput. Phys. 163, 1-21 (2000).

[163] Sapiro, G., *Geometric Partial Differential Equations and Image Processing*, Cambridge University Press, New York, January 2001.

[164] Serra, J., *Image Analysis and Mathematical Morphology, vol. 2: Theoretical Advances*, Academic Press, 1988.

[165] Sethian, J.A., *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Sciences,* Cambridge University Press, Cambridge-UK, 1996.

[166] Sethian, J.A., *A fast marching method for three dimensional photolithography development*, Proc. SPIE 2726, p. 261, (1996).

[167] Shah, J., *A common framework for curve evolution, segmentation, and anisotropic diffusion*, Proc. CVPR, San Francisco, June 1996.

[168] Shi, J., Hu, C. and Shu, C.-W., *A technique of treating negative weights in WENO schemes*, J. Comput. Phys., to appear.

[169] Shu, C.-W., *TVB uniformly high-order schemes for conservation laws*, Math. Comp., 49, 105-121 (1987).

[170] Shu, C.-W., *Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws*, in *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, B. Cockburn, C. Johnson, C.-W. Shu and E. Tadmor (Editor: A. Quarteroni), Lecture Notes in Mathematics, volume 1697, Springer, 1998, pp.325-432.

[171] Shu, C.-W. and Osher, S., *Efficient implementation of essentially non-oscillatory shock capturing schemes*, J. Comput. Phys. 77, 439-471 (1988).

[172] Shu, C.-W. and Osher, S., *Efficient implementation of essentially non-oscillatory shock capturing schemes II*, J. Comput. Phys. 83, 32-78 (1989).

[173] Shu, C.-W., Zang, T.A., Erlebacher, G., Whitaker, D. and Osher, S., *High-order ENO schemes applied to two- and three-dimensional compressible flow*, Appl. Numer. Math. 9, 45-71 (1992).

[174] Simon, L., *Lectures on Geometric Measure Theory*, Australian National University, Australia, 1984.

[175] Steinhoff, J., Fang, M. and Wang, L., *A new Eulerian method for the computation of propagating short acoustic and electromagnetic pulses*, J. Comput. Phys. 157, 683-706 (2000).

[176] Struwe, M., *On the evolution of harmonic mappings of Riemannian surfaces*, Comment. Math. Helvetici 60, 558-581 (1985).

[177] Sussman, M., Smereka, P. and Osher, S., *A level set approach for computing solutions to incompressible two-phase flow*, J. Comput. Phys. 114, 146-159 (1994).

[178] Tang, B., Sapiro, G. and Caselles, V., *Diffusion of general data on non-flat manifolds via harmonic maps theory: The direction diffusion case*, Int. Journal Computer Vision 36:2, 149-161 (2000).

[179] Taubin, G., *Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation*, IEEE Trans. PAMI 13:11, 1115-1138 (1991).

[180] Tek, H and Kimia, B.B., *Image segmentation by reaction-diffusion bubbles*, Proc. ICCV'95, 156-162, Cambridge, June 1995.

[181] Toga, A.W., *Brain Warping*, Academic Press, New York, 1998.

[182] Ton, V.T., Karagozian, A.R., Osher, S. and Engquist, B., *Numerical simulation of high speed chemically reactive flow*, Ther. Comput. Fluid Dyn. 6, 161-179 (1994).

[183] Trygvasson, G., Dahm, W.J.A. and Sbeih, K., *Fine structure of vortex sheet rollup by viscous and inviscid simulation*, J. Fluids Engin. 113, 31-36 (1991).

[184] Tsai, Y.-H., Giga, Y. and Osher, S., *A level set approach for computing discontinuous solutions of Hamilton-Jacobi equations*, to appear in Math. Comp.

[185] Tsitsiklis, J., *Efficient algorithms for globally optimal trajectories*, Proceedings of the 33rd Conference on Decision and Control, Lake Buena Vista, LF, 1368-1373, December 1994.

[186] Tsitsiklis, J., *Efficient algorithms for globally optimal trajectories*, IEEE Transactions on Automatic Control 40, 1528-1538 (1995).

[187] Turing, A., *The chemical basis of morphogenesis*, Philosophical Transactions of the Royal Society B 237, 37-72 (1952).

[188] Turk, G., *Generating textures on arbitrary surfaces using reaction-diffusion*, Computer Graphics 25:4, 289-298 (1991).

[189] van Leer, B., *Towards the ultimate conservative difference scheme V. A second order sequel to Godunov's method*, J. Comput. Phys. 32, 101-136 (1979).

[190] Varah, J., *Stability of difference approximations to the mixed initial boundary value problem for parabolic systems*, SIAM J. Numer. Anal. 8, 598-615 (1971).

[191] Vese, L.A. and Chan, T.F., *A multiphase level set framework for image segmentation using the Mumford and Shah model*, UCLA Math CAM Report 01-25, September 2001

[192] Weickert, J., *Anisotropic Diffusion in Image Processing*, ECMI Series, Teubner-Verlag, Stuttgart, Germany, 1998.

[193] Whitaker, R.T., *Algorithms for implicit deformable models*, Proc. ICCV'95, 822-827, Cambridge, June 1995.

[194] Winkenbach, G. and Salesin, D.H., *Rendering parametric surfaces in pen and ink*, Computer Graphics (SIGGRAPH 96), 469-476 (1996).

[195] Witkin, A.P., *Scale-space filtering*, Int. Joint. Conf. Artificial Intelligence 2, 1019-1021 (1983).

[196] Witkin, A. and Heckbert, P., *Using particles to sample and control implicit surfaces*, Computer Graphics (SIGGRAPH), 269-278 (1994).

[197] Witkin, A. and Kass, M., *Reaction-diffusion textures*, Computer Graphics (SIGGRAPH) 25:4, 299-308 (1991).

[198] Yezzi, A. and Soatto, S., *Stereoscopic segmentation*, IEEE Intl. Conf. on Computer Vision, 59-66, Vancouver, July 2001.

[199] Yngve, G. and Turk, G., *Creating smooth implicit surfaces from polygonal meshes*, Technical Report GIT-GVU-99-42, Graphics, Visualization, and Usability Center. Georgia Institute of Technology, 1999 (obtained from www.cc.gatech.edu/gvu/geometry/publications.html).

[200] Zhang, Y.-T. and Shu, C,-W., *High order WENO schemes for Hamilton-Jacobi equations on triangular meshes*, SIAM J. Sci. Comput., submitted.

[201] Zhao, H.-K., Chan, T., Merriman, B. and Osher, S., *A variational level set approach to multiphase motion*, J. Comput. Phys. 127, 179-195 (1996).

[202] Zhao, H.-K., Osher, S., Merriman, B. and Kang, M., *Implicit nonparametric shape reconstruction from unorganized points using a variational level set method*, Comput. Vision Image Understanding 80, 295-314 (2000).

[203] Zhao, H.-K., Merriman, B., Osher, S. and Wang, C., *Capturing the behavior of bubbles and drops using the variational level set approach*, J. Comput. Phys. 143, p. 495 (1998).

Module 2

Level Set Applications I

Fast Surface Reconstruction Using the Level Set Method

Hong-Kai Zhao* Stanley Osher† Ronald Fedkiw‡

Abstract

In this paper we describe new formulations and develop fast algorithms for implicit surface reconstruction based on variational and partial differential equation (PDE) methods. In particular we use the level set method and fast sweeping and tagging methods to reconstruct surfaces from scattered data set. The data set might consist of points, curves and/or surface patches. A weighted minimal surface-like model is constructed and its variational level set formulation is implemented with optimal efficiency. The reconstructed surface is smoother than piecewise linear and has a natural scaling in the regularization that allows varying flexibility according to the local sampling density. As is usual with the level set method we can handle complicated topologies and deformations, as well as noisy or highly non-unifrom data sets easily. The method is based on a simple rectangular grid, although adaptive and triangular grids are also possible. Some consequences, such as hole filling capability, are demonstrated, as well as a rigorous proof of the viability and convergence of our new fast tagging algorithm.

Keywords: implicit surface, partial differential equations, variational formulation, convection, minimal surface, hole filling

1 Introduction

Surface reconstruction from unorganized data set is very challenging in three and higher dimensions. The problem is ill-posed, i.e, there is no unique solution. Furthermore the ordering or connectivity of data set and the topology of the real surface can be very complicated in three and higher dimensions. A desirable reconstruction procedure should be able to deal with complicated topology and geometry as well as noise and non-uniformity of the data to construct a surface that is a good approximation of the data set and has some smoothness (regularity). Moreover, the reconstructed surface should have a representation and data structure that not only good for static rendering but also good for deformation, animation and other dynamic operation on surfaces. None of the present approaches possess all of these properties. In general there are two kinds of surface representations, explicit or implicit. Explicit surfaces prescribe the precise location of a surface while implicit surfaces represent a surface as a particular isocontour of a scalar function. Popular explicit representations include parametric surfaces and triangulated surfaces. For examples, for parametric surfaces such as NURBS [22, 23], the reconstructed surface is smooth and the data set can be non-uniform. However this requires one to parametrize the data set in a nice way such that the reconstructed surface is a graph in the parameter space. The parametrization and patching can be very difficult for surface reconstruction from an arbitrary data set in three and higher dimensions. Also noise in the data set is difficult to deal with. Another popular approach in computer graphics is to reconstruct a triangulated surfaces using Delaunay triangulations and Voronoi diagrams. The reconstructed surface is typically a subset of the faces of the Delaunay triangulations. A lot of work has been done along these lines [3, 4, 5, 8, 12, 13] and efficient algorithms are available to compute Delaunay triangulations and Voronoi diagrams. Although this approach is more versatile in that it can deal with more general data sets, the constructed surface is only piecewise linear and it is difficult to handle non-uniform and noisy data. Furthermore the tracking of large deformations and topological changes is usually quite difficult using explicit surfaces.

Recently, implicit surfaces or volumetric representations have attracted a lot of attention. There are two main approaches for creating and analyzing implicit surfaces. The traditional approach [7, 18, 27, 29] uses a combination of smooth basis functions, such as blobs, to find a scalar function such that all data points are close to an isocontour of that scalar function. This isocontour represents the constructed implicit surface. Although the implicit surface is usually smooth, the construction is global, i.e. all the basis functions are coupled together and a single data point change can result in globally different coefficients. This makes human interaction, incremental updates and deformation difficult. The second approach uses the data set to define a signed distance function on rectangular grids and denotes the zero isocontour of the signed distance function as the reconstructed implicit surface [6, 9, 16]. The construction of the signed distance function uses a discrete approach and needs an estimation of local tangent planes or normals for the orientation, i.e. a distinction needs to be made between inside and outside. Similar ideas have been applied to shape reconstruction from range data and image fusion [11, 15] where partial connections are available on each piece of data and some "zippering" is needed to patch things together. The advantages of implicit surfaces include topological flexibility, a simple data structure, depth/volumetric information and memory storage efficiency. Using the signed distance representation, many surface operations such as Boolean operations, ray tracing and offset become quite simple. Moreover an extremely efficient marching cubes algorithm [17] is available to turn an implicit surface into a triangulated surface.

We approach this fundamental problem on the continuous level by constructing continuous models using differential geometry and partial differential equations. We also develop efficient and robust numerical algorithms for our continuous formulations. Moreover we combine the level set method and implicit surfaces to provide a general framework for surface modeling, analysis, deformation and many other applications. In our previous work [31] we proposed a new "weighted" minimal surface model based on variational formulations and PDE methods. Only the unsigned distance function to the data set was used in our formulation. Our reconstructed surface is smoother than piecewise linear. In addition, in our formulation there is a regularization that is adaptive to the local sampling density which can keep sharp features if the a local sampling condition is satisfied. The formulation handles noisy as well as non-unform data and works in any number of dimensions. We use the level set method as the numerical technique to deform the implicit surface continuously following the gradient descent of the

*Department of Mathematics, University of California, Irvine, CA 92697-3875, Research supported by NSF DMS 9706566. zhao@math.uci.edu

†Department of Mathematics, University of California, Los Angeles, CA 90095-1555, Research supported by ONR N00014-97-1-0027 and NSF DMS 9706827. sjo@math.ucla.edu

‡Stanford University, Gates Computer Science Bldg., Stanford, CA 94305-9020, Research supported by ONR N00014-97-1-0027. fedkiw@cs.stanford.edu

energy functional for the final reconstruction. Instead of tracking a parametrized explicit surface we solve an PDE on a simple rectangular grid and handle topological changes easily. In this paper we develop a simple physically motivated convection model and a fast tagging algorithm to construct a good initial approximation for our minimal surface reconstruction. This will speed up our previous reconstruction by an order of magnitude. We also introduce a smoothing algorithm similar to [28] as a post process to smooth implicit surfaces or reconstructed implicit surfaces from noisy data.

In the next section we briefly review the variational formulation for the weighted minimal surface model in introduced in [31]. A physically motivated simple convection model is developed in section 3. In section 4 we introduce the level set method for our problems and a simple denoising/smoothing formulation for implicit surfaces . We explain the details of the numerical implementation and fast algorithms in section 5 and show results in section 6.

2 A Weighted Minimal Surface Model

Let $\mathcal{S}$ denote a general data set which can include data points, curves or pieces of surfaces. Define $d(\boldsymbol{x}) = dist(\boldsymbol{x}, \mathcal{S})$ to be the distance function to $\mathcal{S}$. (We shall use bold faced characters to denote vectors.) In [31] the following surface energy is defined for the variational formulation:

$$E(\boldsymbol{\Gamma}) = \left[\int_{\boldsymbol{\Gamma}} d^p(\boldsymbol{x}) ds\right]^{\frac{1}{p}}. \quad 1 \le p \le \infty, \tag{1}$$

where $\boldsymbol{\Gamma}$ is an arbitrary surface and ds is the surface area. The energy functional is independent of parametrization and is invariant under rotation and translation. When $p = \infty$, $E(\boldsymbol{\Gamma})$ is the value of the distance of the point $\boldsymbol{x}$ on $\boldsymbol{\Gamma}$ furthest from $\mathcal{S}$. For $p < \infty$, The surface energy $E(\boldsymbol{\Gamma})$ is equivalent to $\int_{\boldsymbol{\Gamma}} d^p(\boldsymbol{x}) ds$, the surface area weighted by some power of the distance function. We take the local minimizer of our energy functional, which mimics a weighted minimal surface or an elastic membrane attached to the data set, to be the reconstructed surface.

As derived in [31] the gradient flow of the energy functional (1) is

$$\frac{d\boldsymbol{\Gamma}}{dt} = -\left[\int_{\boldsymbol{\Gamma}} d^p(\boldsymbol{x}) ds\right]^{\frac{1}{p}-1} d^{p-1}(\boldsymbol{x}) \left[\nabla d(\boldsymbol{x}) \cdot \boldsymbol{n} + \frac{1}{p} d(\boldsymbol{x}) \kappa\right] \boldsymbol{n}, \tag{2}$$

and the minimizer or steady state solution of the gradient flow satisfies the Euler-Lagrange equation

$$d^{p-1}(\boldsymbol{x}) \left[\nabla d(\boldsymbol{x}) \cdot \boldsymbol{n} + \frac{1}{p} d(\boldsymbol{x}) \kappa\right] = 0, \tag{3}$$

where $\boldsymbol{n}$ is the unit outward normal and κ is the mean curvature. We see a balance between the attraction $\nabla d(\boldsymbol{x}) \cdot \boldsymbol{n}$ and the surface tension $d(\boldsymbol{x})\kappa$ in the equations above. Moreover the nonlinear regularization due to surface tension has a desirable scaling $d(\boldsymbol{x})$. Thus the reconstructed surface is more flexible in the region where sampling density is high and is more rigid in the region where the sampling density is low. In the steady state equation(3) above, since $\nabla d \cdot \boldsymbol{n} \le 1$, a local sampling density condition similar to the one proposed in [4], which says sampling densities should be proportional to the local curvature of the feature. To construct the minimal surface we used a continuous deformation in [31]. We start with an initial surface that encloses all data and follow the gradient flow (2). The parameter p affects the flexibility of the membrane to some extent. When $p = 1$, the gradient flow (2) is scale invariant i.e., dimensionless. In practice we find that $p = 1$ or 2 (similar to a least squares formulation) are good choices. Some more details can be found in [31].

In two dimensions, it was shown in [31] that a polygon which connects adjacent points by straight lines is a local minimum. This result shows a connection between the variational formulation and previous approaches. On the other hand this result is not surprising since a minimal surface passing through two points is a straight line in two dimensions. However in three dimensions the situation becomes much more interesting. The reconstructed minimal surface has no edges and is smoother than a polyhedron.

3 The Convection Model

The evolution equation (2) involves the mean curvature of the surface and is a nonlinear parabolic equation. A time implicit scheme is not currently available. A stable time explicit scheme requires a restrictive time step size, $\Delta t = O(h^2)$, where h is the spatial grid cell size. Thus it is very desirable to have an efficient algorithm to find a good approximation before we start the gradient flow for the minimal surface. We propose the following physically motivated convection model for this purpose.

The convection of a flexible surface $\boldsymbol{\Gamma}$ in a velocity field $\boldsymbol{v}(\boldsymbol{x})$ is described by the differential equation

$$\frac{d\boldsymbol{\Gamma}(t)}{dt} = \boldsymbol{v}(\boldsymbol{\Gamma}(t)).$$

If the velocity field is created by a potential field $\mathcal{F}$, then $\boldsymbol{v} = -\nabla \mathcal{F}$. In our convection model the potential field is the distance function $d(\boldsymbol{x})$ to the data set $\mathcal{S}$. This leads to the convection equation

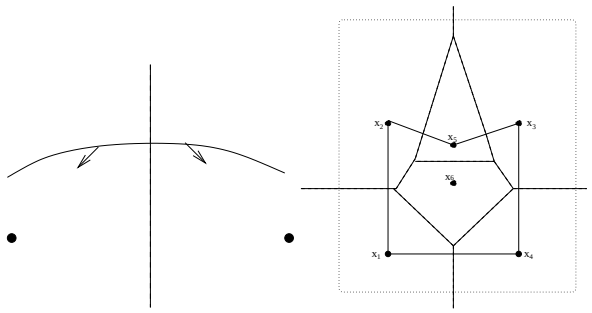
$$\frac{d\boldsymbol{\Gamma}(t)}{dt} = -\nabla d(\boldsymbol{x}). \tag{4}$$

For example, if the data set contains a single point $\boldsymbol{x}_0$, the potential field is $d(\boldsymbol{x})$=$|\boldsymbol{x}-\boldsymbol{x}_0|$ and the velocity field is $\boldsymbol{v}(\boldsymbol{x}) = -\nabla d(\boldsymbol{x}) = -\frac{\boldsymbol{x}-\boldsymbol{x}_0}{|\boldsymbol{x}-\boldsymbol{x}_0|}$, a unit vector pointing towards $\boldsymbol{x}_0$. Any particle in this potential field will be attracted toward $\boldsymbol{x}_0$ along a straight line with unit speed. For a general data set $\mathcal{S}$, a particle will be attracted to its closest point in $\mathcal{S}$ unless the particle is located an equal distance from two or more data points. The set of equal distance points has measure zero. Similarly, points on a curve or a surface, except those equal distance points, are attracted by their closest points in the data set (see Fig. 1(a)). The ambiguity at those equal distance points is resolved by adding a small surface tension force which automatically exists as numerical viscosity in our finite difference schemes. Those equal distance points on the curve or surface are dragged by their neighbors and the whole curve or surface is attracted to the data set until it reaches a local equilibrium (see Fig.1(b)), which is a polygon or polyhedron whose vertices belong to the data set as the viscosity tends to zero (see Fig.1(b)).

Here are some properties of this simple convection model: (1) the normal velocity of the curve or the surface is less than or equal to 1, (2) each point of the curve or surface is attracted by its closest point in the data set.

Figure 1(b) is an illustration of the convection of a curve. The initial curve (the dotted rectangle) feels the attraction of $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4$ and closes in. Then it begins to feel $\boldsymbol{x}_5$. The final shape is a pentagon that goes through $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4$ and $\boldsymbol{x}_5$ while $\boldsymbol{x}_6$ is screened out.

Since the convection equation is a first order linear differential equation, we can solve it using a time step $\Delta t = O(h)$ leading to significant computational savings over typical parabolic $\Delta t = O(h^2)$ time step restrictions. The convection model by itself very often results in a good surface reconstruction. In section 5 we will construct a very fast tagging algorithm that finds a crude approximation of the local equilibrium solution for our convection model.



(a) (b)

(a) the attraction of a piece of curve by two points, (b) dotted line is the initial curve, solid line is the final curve, dashed line is the Voronoi diagram.

Figure 1:

4 The Level Set Formulation

In general we do not have any á priori knowledge about the topology of the shape to be reconstructed. Topological changes may occur during the continuous deformation process. This makes explicit tracking, which requires consistent parametrization, almost impossible to implement. Here we introduce the level set method as a powerful numerical technique for the deformation of implicit surfaces. Although implicit surfaces have been used in computer graphics for quite a while, they were mostly used for static modeling and rendering and were based on discrete formulations [7]. The *level set method* is based on a continuous formulation using PDEs and allows one to deform an implicit surface, which is usually the zero isocontour of a scalar (level set) function, according to various laws of motion depending on geometry, external forces, or a desired energy minimization. In numerical computations, instead of explicitly tracking a moving surface we implicitly capture it by solving a PDE for the level set function on rectangular grids. The data structure is extremely simple and topological changes are handled easily. The level set formulation works in any number of dimensions and the computation can easily be restricted to a narrow band near the zero level set, see e.g. [1, 21]. We can locate or render the moving surface easily by interpolating the zero isosurface of the level set function. The level set method was originally introduced by Osher and Sethian in [20] to capture moving interfaces and has been used quite successfully in moving interface and free boundary problems as well as in image processing, image segmentation and elsewhere. See [19] for a comprehensive review.

Two key steps for the level set method are:

- Embed the surface: we initially represent a co-dimension one surface $\mathbf{\Gamma}$ as the zero isocontour of a scalar (level set) function $\phi(\boldsymbol{x})$, i.e. $\mathbf{\Gamma} = \{\boldsymbol{x} : \phi(\boldsymbol{x}) = 0\}$. $\phi(\boldsymbol{x})$ is negative inside $\mathbf{\Gamma}$ and positive outside $\mathbf{\Gamma}$. Geometric properties of the surface $\mathbf{\Gamma}$, such as the normal, surface area, volume, mean and Gaussian curvature can be easily computed using ϕ. For example, the outward unit normal $\boldsymbol{n}$ is simply $\frac{\nabla\phi}{|\nabla\phi|}$ and the mean curvature κ is $\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}$.

- Embed the motion: we derive the time evolution PDE for the level set function such that the zero level set has the same motion law as the moving surface, i.e. the moving surface coincides with the zero level set for all time. Since $\mathbf{\Gamma}(t) = \{\boldsymbol{x} : \phi(\boldsymbol{x}, t) = 0\}$,

$$\frac{d\phi(\mathbf{\Gamma}(t), t)}{dt} = \phi_t + \frac{d\mathbf{\Gamma}(t)}{dt} \cdot \nabla\phi = 0, \tag{5}$$

where we replace $\frac{d\mathbf{\Gamma}(t)}{dt}$ with a velocity field $\boldsymbol{v}(\boldsymbol{x})$ defined for all $\boldsymbol{x}$ and equal to $\frac{d\mathbf{\Gamma}(t)}{dt}$ for $\boldsymbol{x}$ on $\mathbf{\Gamma} = \{\boldsymbol{x} : \phi(\boldsymbol{x}, t) = 0\}$.

To develop the time evolution PDE for the level set function, one needs to extend the velocity at the zero level set, which is given by the motion law of the original surface, to other level sets in a natural way. For geometric motions, i.e. where the motion law (velocity) depends only on the geometry of the moving surface, the most natural way to define $\boldsymbol{v}$ is to apply the same motion law for all level sets of the level set function, which will result in a morphological PDE [2]. For example, the gradient flow (2) is a geometric motion. Using the fact (see, e.g., [25, 31])

$$\int_{\Gamma} d^p(\boldsymbol{x}) ds = \int d^p(\boldsymbol{x}) \delta(\phi(\boldsymbol{x})) |\nabla\phi(\boldsymbol{x})| d\boldsymbol{x},$$

where $\phi(\boldsymbol{x}, t)$ is the level set function whose zero level set is $\mathbf{\Gamma}(t)$ and $\delta(x)$ is the one dimensional delta function, and extending the motion (normal velocity) to all level sets we have the level set formulation for the gradient flow (2)

$$\frac{\partial\phi}{\partial t} = \frac{1}{p}|\nabla\phi| \left[\int d^p(\boldsymbol{x})\delta(\phi)|\nabla\phi| d\boldsymbol{x}\right]^{\frac{1}{p}-1} \nabla\cdot\left[d^p(\boldsymbol{x})\frac{\nabla\phi}{|\nabla\phi|}\right], \tag{6}$$

For the convection model (4), since the velocity field $-\nabla d(\boldsymbol{x})$ is defined everywhere, we can naturally extend the convection to all level sets of $\phi(\boldsymbol{x}, t)$ to obtain

$$\frac{\partial\phi}{\partial t} = \nabla d(\boldsymbol{x}) \cdot \nabla\phi. \tag{7}$$

Although all level set functions are equally good theoretically, in practice the signed distance function is preferred for numerical computations. However even if we start with a signed distance function the level set function will generally not remain a signed distance function. As an example, in the convection model all level sets are attracted to the data set simultaneously and they become more and more packed together. We need a procedure to force them apart while keeping the zero level set intact. We use a numerical procedure called reinitialization, see e.g. [21, 25], to redistance the level set function locally without interfering with the motion of the zero level set. The reinitialization process will also provide us with a signed distance function for rendering the implicit surface after the deformation procedure stops.

If the data set contains noise, we derive a post-smoothing process similar to that of [28] for our reconstructed implicit surfaces using the variational level set formulation. Let ϕ_0 denote the initial level set function whose zero level set is the surface we would like to denoise or smooth. We define the denoised or smoothed implicit surface as the zero level set of ϕ that minimizes the following functional

$$\frac{1}{2}\int (H(\phi) - H(\phi_0))^2 d\boldsymbol{x} + \epsilon \int \delta(\phi)|\nabla\phi| d\boldsymbol{x}, \tag{8}$$

where $H(x)$ is the one dimensional Heaviside function. The first term in the above energy functional is a fidelity term that measures the symmetric volume difference between two closed surfaces. The second integral in the above functional is the surface area of the zero level set of ϕ, which is a regularization term that minimizes the surface area of the denoised or smoothed surface. The constant ϵ is a parameter that controls the balance between the fidelity and the regularization. We again find the minimizer by following the gradient flow of (8), whose level set formulation is:

$$\phi_t = |\nabla\phi|[\epsilon\kappa - (H(\phi) - H(\phi_0))]$$

To some extent this variational formulation is also related to Total Variation (TV) denoising for images proposed in [24]. In fact it is exactly TV denoising applied to $H(\phi)$, since the total variation of a function can be represented as the integration of the parameter length of all level sets of the function by co-area formula [14].

5 Numerical Implementation

There are three key numerical ingredients in our implicit surface reconstruction. First, we need a fast algorithm to compute the distance function to an arbitrary data set on rectangular grids. Second, we need to find a good initial surface for our gradient flow. Third, we have to solve time dependent PDEs for the level set function.

5.1 Computing the distance function

The distance function $d(\boldsymbol{x})$ to an arbitrary data set $\mathcal{S}$ solves the following Eikonal equation:

$$|\nabla d(\boldsymbol{x})| = 1, \qquad d(\boldsymbol{x}) = 0,\ \boldsymbol{x} \in \mathcal{S}. \tag{9}$$

From the PDE point of view, the characteristics of this Eikonal equation are straight lines which radiate from the data set. This reveals the causality property for the solution of the PDE, i.e., the information propagates along straight lines from the data set, and the solution at a grid point should be determined only by its neighboring grid points that have smaller distance values. We use an algorithm [10, 31] that combines upwind differencing with Gauss-Seidel iterations of different sweeping order to solve (9) on rectangular grids. From numerical experiments it seems that the total number of iterations is independent of mesh size, i.e. the complexity is $O(M+N)$ for N grid points and M data points.

Suppose we have a set of data points and a rectangular grid. We use an initialization procedure, of complexity $O(M+N)$ to assign initial values for N grid points and M data points. Those gird points that belong to the data set are assigned zero. Those grid points that are neighbors (i.e., vertices of grid cells that contain data points,) are assigned the exact distance values. These grid points are our boundary points and their distance values will not change in later computations. We assign a large positive number to all other grid points. These values will be updated in later computations. We can deal with more general data set as long as the distance values on grids neighboring to the set are provided initially. In one dimension, the following upwind differencing is used to discretize the Eikonal equation (9) at ith grid point that are not boundary points,

$$[(d_i - d_{i-1})^+]^2 + [(d_i - d_{i+1})^+]^2 = h^2, \ \ i = 1, 2, \ldots, I \tag{10}$$

where h is the grid size, I is the total number of grids and $(x)^+ = \begin{cases} x & x > 0 \\ 0 & x \le 0 \end{cases}$. We use two different sweeps of Gauss-Seidel iterations successively, i.e., for $i = 1 : I$ and $i = I : 1$, to solve this system of equations. At the ith grid, using the current values of d_{i-1} and d_i, there exists at least one solution for equation (10) $[min(d_{i-1}, d_{i+1}) + \frac{h}{\sqrt{2}}, max(d_{i-1}, d_{i+1}) + \frac{h}{\sqrt{2}}]$, which only depends on neighbors with smaller values. We take d_i to be the smaller one if there are two solutions. It can be shown that these two sweeps will get the exact solution of the discrete system (10), which is of first order $O(h)$ accuracy to the real distance function. In two dimensions, a slightly more complicated system,

$$\begin{aligned}&[(d_{i,j} - d_{i-1,j})^+]^2 + [(d_{i,j} - d_{i+1,j})^+]^2 \\ &+[(d_{i,j} - d_{i,j-1})^+]^2 + [(d_{i,j} - d_{i,j+1})^+]^2 = h^2,\end{aligned}$$

$i = 1, \ldots, I,\ j = 1, \ldots, J$, has to be solved using sweeps of Gauss-Seidel iterations of four different orders,

$$\begin{aligned}&(1)\ i = 1 : I, j = 1 : J \quad (2)\ i = 1 : I, j = J : 1 \\ &(3)\ i = I : 1, j = 1 : J \quad (4)\ i = I : 1, j = J : 1\end{aligned}$$

In most numerical computations, a total of five or six sweeps is enough in two dimensions. Similarly a three dimensional extension is straight forward. This distance algorithm is versatile, efficient and will be used in later stages of the surface reconstruction.

5.2 Finding a good initial guess

We can use an arbitrary initial surface that contains the data set such as a rectangular bounding box, since we do not have to assume any á priori knowledge for the topology of the reconstructed surface. However, a good initial surface is important for the efficiency of our PDE based method. On a rectangular grid, we view an implicit surface as an interface that separates the exterior grid points from the interior grid points. In other words, volumetric rendering requires identifying all exterior (interior) grid points correctly. Based on this idea, we propose a novel, extremely efficient tagging algorithm that can identify as many correct exterior grid points as possible and hence provide a good initial implicit surface. As always, we start from any initial exterior region that is a subset of the true exterior region. Here is the description of our fast tagging algorithm and the proof of its viability. For simplicity of exposition only, we shall consider a uniform grid

$$(x_i, y_j) = \boldsymbol{x}_{ij} = (i\Delta, j\Delta), \ \ i, j = 0, \pm 1, \pm 2, -$$

in 2 dimensions, where Δ is the grid size. The results work in any number of dimensions and for more general grid structure.

Let $d_{ij} = d(\boldsymbol{x}_{ij})$ be the unsigned distance of $\boldsymbol{x}_{ij}$ to the data set $\mathcal{S}$ (i.e. to the closest point on $\mathcal{S}$). We say $\boldsymbol{x}_{ij} < \boldsymbol{x}_{kl}$, or $\boldsymbol{x}_{ij}$ is closer than $\boldsymbol{x}_{kl}$ or $\boldsymbol{x}_{ij}$ is smaller than $\boldsymbol{x}_{kl}$ if $d_{ij} < d_{kl}$.

We define $\mathcal{S}^\Delta$ to be the set of grid nodes $\boldsymbol{x}_{ij}$ for which $d_{ij} < \Delta$. (Note: if every data point lies on a grid node, then $\mathcal{S}^\Delta = \mathcal{S}$).

We wish to obtain a set Ω for which $\mathcal{S}^\Delta \subset \Omega$ and for which the boundary $\partial\Omega$ serves as a very good initial guess for our final reconstructed surface. From our convection model, we need to rapidly find a crude approximate solution to the steady state equation

$$\nabla d(x) \cdot \boldsymbol{n} = 0,$$

where $\boldsymbol{n}$ is the unit outward normal of the boundary $\partial\Omega$. This equation can be written (in 2D) as

$$d_x \phi_x + d_y \phi_y = 0, \tag{11}$$

where ϕ is the level set function whose zero level set is $\partial\Omega$ that surrounds $\mathcal{S}^\Delta$. We wish to march quickly in a manner reminiscent of the fast algorithm of [26], but for a very different problem – this is not the eikonal equation, and steady states generally depend on the initial guess. There are some similarities in that a heap sort algorithm is used as is the Cartesian structure of the grid.

For a point $\boldsymbol{x}_{ij}$ we define its neighbors as the four points $\boldsymbol{x}_{i\pm1,j}, \boldsymbol{x}_{i,j\pm1}$. A boundary point of a set Ω is defined to be the set of $\boldsymbol{x}_{ij}$ in Ω for which at least one of its four neighbors are in the exterior of Ω. The boundary of Ω is denoted by $\partial\Omega$.

Given Ω_0 for which $\mathcal{S}^\Delta$ we shall march quickly towards $\Omega \subset \Omega_0$ whose boundary $\partial\Omega$ will act as our initial level surface for the convection and convection-diffusion algorithms defined below. This is our fast tagging algorithm.

We begin by considering $\partial\Omega_0$ which we order in a nondecreasing sequence via a heap sort algorithm. We denote $\partial\Omega_0$ as a temporary boundary set at stage 0. We shall inclusively create Ω_n and a temporary boundary set $\partial\Omega_{n,t} \subset \partial\Omega_n$ so that, after a finite number of steps the largest point in $\partial\Omega_{n,t}$ is also in $\mathcal{S}^\Delta$, at which point the algorithm terminates and Ω_n is the final Ω. At each marching step, we either tag the largest (furthest) temporary boundary point into the final boundary or turn the largest (furthest) temporary boundary

point into an exterior point. This fast tagging algorithm is of complexity $O(N \log N)$; the log N term appears because of the sorting step.

The tagging algorithm is as follows: Consider the largest $\boldsymbol{x}_{ij}^{(n)} \in \partial\Omega_{n,t}$.

(a) If there is at least one interior neighbor of $\boldsymbol{x}_{ij}^{(n)}$ which is not closer to $\mathcal{S}$, put $\boldsymbol{x}_{ij}^{(n)}$ into the tagged boundary set and define $\partial\Omega_{n+1,t} = \partial\Omega_{n,t} - \{\boldsymbol{x}_{ij}^{(n)}\}, \Omega_{n+1} = \Omega_n$.

(b) If all interior neighbors of $\boldsymbol{x}_{ij}^{(n)}$ are closer to $\mathcal{S}$, put $\boldsymbol{x}_{ij}^{(n)}$ into the exterior and include its interior neighbors into the new temporary boundary, i.e., define $\Omega_{n+1} = \Omega_n - \{\boldsymbol{x}_{ij}^{n}\}$ and $\partial\Omega_{n+1,t} = \partial\Omega_{n+1} - \{\partial\Omega_n - \partial\Omega_{n,t}\}$.

Repeat this process until either (a) the temporary boundry set becomes empty, or (b) maximum distance of the untagged temporary boundary points, (the set $\partial\Omega_{n,t}$) to $\mathcal{S}$ is less than Δ.

We now prove the algorithm is viable and converges. If condition (a) is satisfied at stage n, then since $\partial\Omega_{n+1,t} \subset \partial\Omega_{n,t}$ $\boldsymbol{x}_{ij}^{(n+1)} \leq \boldsymbol{x}_{ij}^{(n)}$. If condition (b) is satisfied then $\partial\Omega_{n+1,t}$ will include new points that are neighbors of $\boldsymbol{x}_{ij}^{(n)}$ and are closer to $\mathcal{S}$ than $\boldsymbol{x}_{ij}^{(n)}$. Thus $\boldsymbol{x}_{ij}^{(n+1)} \leq \boldsymbol{x}_{ij}^{(n)}$. This ends the proof that the algorithm is viable and converges.

Remark1: Our tagging algorithm produces a very crude approximation to the steady state solution of the convection equation, which we rewrite: $\nabla d \cdot \nabla \phi = 0$. We solve this crudely on a grid for a function $\phi_i j$ which has value either +1 or -1. We initialize so that $\phi_i j = 1$ in the exterior of $\Omega_0, \phi_i j = -1$ inside Ω_0. At every grid point $\boldsymbol{x}_i j$ to be updated we march in an "upwind" direction, which means the new $\phi_i j$ depends only on values at the four neighbors which are further from $\mathcal{S}$ than $\boldsymbol{x}_i j$. Thus we order the temporary boundary points and update the largest untagged point via a crude process

$$\phi_{ij} = P[\phi^n_{i-1,j}, \phi^n_{i+1,j}, \phi^n_{i,j-1}, \phi^n_{i,j}]$$

where P denotes a procedure that picks one of the values from its arguments that corresponds to a more remote point as follows: if there are any more remote interior points, it picks the one which is furthest from $\mathcal{S}$. Else it picks the furthest exterior point. This is equivalent to using a convex combination of either interior or exterior points to approximate ∇d and enforce ϕ to be constant along ∇d It is easy to see that this approximates the level set equation (11) and is exactly our tagging algorithm.

Remark2: At every stage of our tagging algorithm, all points $x^n_{\nu\mu}$ which are interior points of Ω_n and which are more remote than the point being tagged, x^n_{ij}, will remain in Ω_N for all $N > n$, and hence in the final set Ω, since the maximum distance on the untagged temporary boundary is decreasing. Thus we generally obtain a nontrivial limit set Ω.

Figure 2 illustrates how our fast tagging algorithm works. Starting from an arbitrary exterior region that is a subset of the final exterior region, the furthest point on the temporary boundary is tangent to a distance contour and does not have an interior point that is farther away. The furthest point will be tagged as an exterior point and the boundary will move inward at that point. Now another point on the temporary boundary becomes the furthest point and hence the whole temporary boundary moves inward. After a while the temporary boundary is close to a distance contour and moves closer and closer to the data set following the distance contours until the distance contours begin to break into spheres (circles in the 2D figure) around data points. We now see that the temporary boundary point at the breaking point of the distance contour, which is equally distant from distinct data points, will have neighboring interior points that have a larger distance. So this temporary boundary point will be tagged as a final boundary point by our procedure and the temporary boundary will stop moving inward at this breaking point. The temporary boundary starts deviating from the distance contours and continues moving closer to the data set until all temporary boundary points either have been tagged as final boundary points or are close to the data points. The final boundary is approximately a a polyhedron (polygon in 2D) with vertices belonging to the data set.

This general tagging algorithm can incorporate human interaction easily by putting any new exterior point(s) or region(s) into our tagged exterior region at any stage in our tagging algorithm. After the tagging algorithm is finished we again use the fast distance algorithm to compute a signed distance to the tagged final boundary.

The tagging method above requires an initial guess for the exterior region. This can either be the bounding box of our computational rectangular domain or an outer contour of the distance function, $d(\boldsymbol{x}) = \epsilon$. An outer contour of the distance function can be found by starting with any exterior point, such as the corners of of our rectangular domain, and expanding the exterior region by repeatedly tagging those grid points which are connected to the starting exterior point and have a distance larger than ϵ as exterior points. When the tagging algorithm is finished the boundary of the exterior region is approximately the outer contour of $d(\boldsymbol{x}) = \epsilon$ or roughly an ϵ offset of the real shape. When using this $d(\boldsymbol{x}) = \epsilon$ contour, first proposed in [31], one needs to exercise caution in choosing ϵ. For example, if ϵ is too small, we will have isolated spheres surrounding data points. If the sampling density of the data set is fine enough to resolve the real surface, then we can find an appropriate ϵ and get a very good initial surface with $O(N+M)$ operations. When combined with the above fast tagging algorithm, we can find a good initial approximation very efficiently. For non-uniform data points the intersection of a bounding box and a distance contour with moderate ϵ, which is a simple Boolean operation for implicit surfaces, often gives a good initial surface.

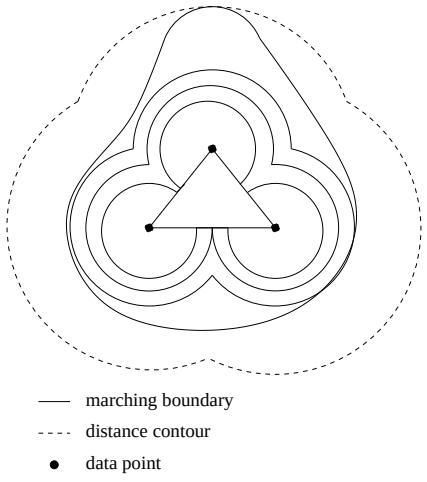


Figure 2:

5.3 Solving the partial differential equation.

After we find the distance function $d(\boldsymbol{x})$ and a good initial implicit surface using the above algorithms, we can start the continuous deformation following either the gradient flow (2) or the convection (4) using the corresponding level set formulation (6) or (7). Our numerical implementations are based on standard algorithms for the level set method. The one dimensional Delta function $\delta(x)$ and Heaviside function $H(x)$ are approximated numerically if needed.

Details can be found in, for example, [21, 30, 31]. The convection model is simple and fast but the reconstructed surface is close to a piecewise linear approximation. In contrast, the energy minimizing gradient flow, which contains a weighted curvature regularization effect, is more complicated and computationally expensive but reconstructs a smooth weighted minimal surface. These two continuous deformations can be combined, and in particular, the gradient flow can be used as a smoothing process for implicit surfaces. In most of our applications, about one hundred time steps in total are enough for our continuous deformation. Since we use a reinitialization procedure regularly during the deformation, we finish with a signed distance function for the reconstructed implicit surface.

5.4 Multiresolution

There are two scales in our surface reconstruction. One is the resolution of the data set. The other is the resolution of the grid. The computational cost generally depends mainly on the grid size. To achieve the best results those two resolutions should be comparable. However our grid resolution can be independent of the sampling density. For example, we can use a low resolution grid when there is noise and redundancy in the data set or when memory and speed are important. From our numerical results figure 9(c) our reconstruction is quite smooth even on a very low resolution grid. We can also use a multiresolution algorithm, i.e., reconstruct the surface first on coarser grids and interpolate the result to a finer resolution grid for further refinement in an hierarchical way.

5.5 Efficient storage

To store or render an implicit surface, we only need to record the values and locations (indices) of those grid points that are next to the surface, i.e., those grid points that have a different sign from at least one of their neighbors. These grid points form a thin grid shell surrounding the implicit surface. No connectivity or other information needs to be stored. We reduce the filesize by at least an order of magnitude by using this method. Moreover we can easily reconstruct the signed distance function in $O(N)$ operations for the implicit surface using the following procedure. (1) Use the fast distance finding algorithm to find the distance function using the absolute value of the stored grid shell as an initial condition. (2) Use a tagging algorithm, similar to the one used above to find exterior points outside a distance contour, to identify all exterior points and interior points separated by the stored grid shell and turn the computed distance into the signed distance. For example, if we store the signed distance funtion for our reconstructed Happy Buddha from almost half a million points on a $146 \times 350 \times 146$ grid in binary form, the file size is about 30MB. If we use the above efficient way of storage the file size is reduced to 2.5MB without using any compression procedure and we can reconstruct the signed distance function in 1 minute using the above algorithm .

6 Results

In this section we present a few numerical examples that illustrate the efficiency and quality of our surface construction. In particular we show (1) how the level set method handles surface deformation and topological change easily, (2) how quickly our tagging algorithm constructs a good initial guess, (3) how smooth the reconstructed surfaces are by using either the convection model or the minimal surface model, (4) how our algorithm works with non-uniform, noisy or damaged data, and (5) how multiresolution works in our formulation. All calculations were done with a Pentium III, 600Mhz processor. Data points for the drill, the dragon and the Buddha were obtained from www-graphics.stanford.edu/data/3Dscanrep and data points for the hand skeleton and turbine blade were obtained from www.cc.gatech.edu/projects/large_models. Only locations of the data points are used in our reconstructions.

The first group of examples show surface reconstruction from synthesized data. Figure 4 show surface reconstruction, a torus, from damaged data, which is like hole filling. Figure 5 shows the reconstruction of a sphere from a box using eight longitudinal circles and eight latitudinal circles. For this example we do not have any discrete data points. We only provide the unsigned distance function. This can also be viewed as an extreme case of non-uniform data. Figure 5(a) shows those circles and figure 5(b) shows reconstruction using the convection model. Figure 5(c) shows the final minimal surface reconstruction following the gradient flow on top of figure 5(b).

The second group of examples are from real data. Timings, number of data points and grid size are shown in table 3. CPU time is measured in minutes. CPU (initial) is the time for the initial reconstruction using the distance contour and the fast tagging algorithm. CPU (total) is the total time used for the reconstruction. Since our PDE based algorithms are iterative procedures, different convergence criterion will give different convergence times. For data sets that are fairly uniform, such as the drill, the dragon, the Buddha and the hand skeleton, we start with an outer distance contour and use the fast tagging algorithm to get an initial reconstruction. The initial reconstruction is extremely fast, as we can see from table 3. After the initial reconstruction, we first use the convection model and then use the gradient flow to finish the final reconstruction. In our reconstruction, the grid resolution is much lower than the data samples and yet we get final results that are comparable to the reconstructions shown at those websites above.

Figure 6 shows the reconstruction for a rat brain from MRI slices. The data set is very non-uniform and noisy. We start with the intersection of a bounding box and an outer distance contour with relatively large $\epsilon = 12h$, which is shown in figure 6(b). The next example, figure 7 is our reconstruction of a 1.6mm drill bit from 1961 scanned data points. It is a quite challenging example for most methods for surface reconstruction from unorganized data as is shown in [11]. Figure 8 shows the reconstruction of a hand skeleton. Figure 9 shows the reconstruction of the Happy Buddha. Figure 9(a) shows the initial reconstruction using the fast tagging algorithm only. We start with an outer distance contour, $d = 3h$, initially and it takes only 3 minutes for half a million points on a $146 \times 350 \times 146$ grid. Figure 10 is the reconstruction of the dragon. Figure 9(b) is the final reconstruction. Figure 9(c) is the reconstruction on a much under resolved coarse grid $63 \times 150 \times 64$ using the same amount of data points. It only takes 7 minutes and the result is quite good. For the example of the dragon, we show the initial reconstruction in figure 10(a), reconstruction using the convection model only in figure 10(b) and the final weighted minimal surface reconstruction in figure 10(c). Figure 10(d) shows the reconstruction using a much lower resolution data set on the same grid and the result is quite comparable to figure 10(c). The final example shows the reconstruction of a turbine blade on a $178 \times 299 \times 139$ grid for almost a million data points.

7 Conclusions

We present a variational and PDE based formulation for surface reconstruction from unorganized data. Our formulation only depends on the (unsigned) distance function to the data and the final reconstruction is smoother than piecewise linear. We use the level set method as a numerical tool to deform and construct implicit surfaces on fixed rectangular grids. We use fast sweeping algorithms for computing the distance function and fast tagging algorithms for

Model	Data points	Grid size	CPU (initial)	CPU (total)
Rat brain	1506	80x77x79	.12	3
Drill	1961	24x250x32	0.1	2
Buddha	543652	146x350x146	3	68
Buddha	543652	63x150x64	.3	7
Dragon	437645	300x212x136	4	77
Dragon	100250	300x212x136	3	66
Hand	327323	200x141x71	.5	10
Turbine blade	882954	178x299x139	2.5	60

Figure 3: timing table

initial construction. Our method works for complicated topology and non uniform or noisy data.

References

[1] D. Adalsteinsson and J.A. Sethian. A fast level set method for propagating interfaces. *J. Comp. Phys.*, 118(2):269–277, 1995.

[2] L. Alvarez, F. Guichard, P.-L. Lions, and J.-M. Morel. Axioms and fundamental equations of image processing. *Arch. Rat. Mechanics*, 123:199–257, 1993.

[3] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *14th ACM Symposium on Computational Geometry*, 1998.

[4] N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60/2(2):125–135, 1998.

[5] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. *Proc. SIGGRAPH'98*, pages 415–421, 1998.

[6] C. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3d scans. *SIGGRAPH'95 Proceedings*, pages 193–198, July 1995.

[7] J. Bloomenthal, C. Bajaj, J. Blinn, M.-P Cani-Gascuel, A. Rockwood, B. Wyvill, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to implicit surfaces.* Morgan Kaufman, Inc., San Francisco, 1997.

[8] J.D. Boissonnat. Geometric structures for three dimensional shape reconstruction. *ACM Trans. Graphics 3*, pages 266–286, 1984.

[9] J.D. Boissonnat and F. Cazals. Smooth shape reconstruction via natural neighbor interpolation of distance functions. *ACM Symposium on Computational Geometry*, 2000.

[10] M. Boué and P. Dupuis. Markov chain approximations for deterministic control problems with affine dynamics and quadratic cost in the control. *SIAM J. Numer. Anal.*, 36(3):667–695, 1999.

[11] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *SIGGRAPH'96 Proceedings*, pages 303–312, 1996.

[12] H. Edelsbrunner. Shape reconstruction with Delaunay complex. In *Proc. of LATIN'98: Theoretical Informatics*, volume 1380 of *Lecture Notes in Computer Science*, pages 119–132. Springer-Verlag, 1998.

[13] H. Edelsbrunner and E. P. Mücke. Three dimensional α shapes. *ACM Trans. Graphics 13*, pages 43–72, 1994.

[14] L.C. Evans and R.F. Gariepy. Measure theory and fine properties of functions. *Studies in Advanced Mathematics, CRC Press Inc.*, 1992.

[15] A. Hilton, A.J. Stoddart, J. Illingworth, and T. Windeatt. Implicit surface - based geometric fusion. *Comput. Vision and Image Understanding*, 69:273–291, 1998.

[16] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH'92 Proceedings*, pages 71–78, 1992.

[17] W.E. Lorenson and H.E. Cline. Marching cubes: A high resolution 3d surface recomstruction algorithm. *Computer Graphics*, 21:163–169, 1987.

[18] S. Muraki. Volumetric shape description of range data using "blobby model". In *Computer Graphics (Proc. SIGGRAPH)*, volume 25, pages 227–235, July 1991.

[19] S. Osher and R. Fedkiw. Level set methods: an overview and some recent results. *to appear in J. Comp. Phys.*, 2000.

[20] S. Osher and J. Sethian. Fronts propagating with curvature dependent speed, algorithms based on a Hamilton-Jacobi formulation. *J. Comp. Phys.*, 79:12–49, 1988.

[21] D. Peng, B. Merriman, S. Osher, H.K. Zhao, and M. Kang. A PDE based fast local level set method. *J. Comp. Phys.*, 155:410–438, 1999.

[22] L. Piegl and W. Tiller. *The NURBS book.* Berlin, Germany: Springer-Verlag, 2nd edition edition, 1996.

[23] D.F. Rogers. *An Introduction to NURBS.* Morgan Kaufmann, 2000.

[24] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[25] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flows. *J. Comp. Phys.*, 119:146–159, 1994.

[26] J.N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40((9)):1528–1538, 1995.

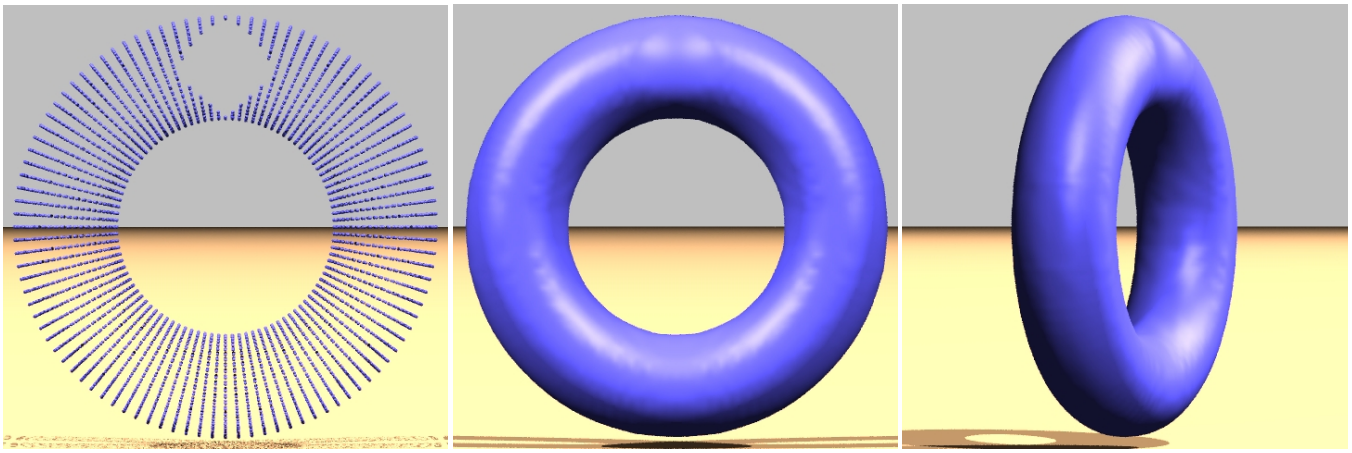
[27] G. Turk and J. ÒBrien. Shape transformation using variational implicit functions. *SIGGRAPH99*, pages 335–342, August 1999.

[28] R. Whitaker. A level set approach to 3D reconstruction from range data. *International journal of Computer Vision*, 1997.

[29] A.P. Witkin and P.S. Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH94)*, pages 269–278, July 1994.

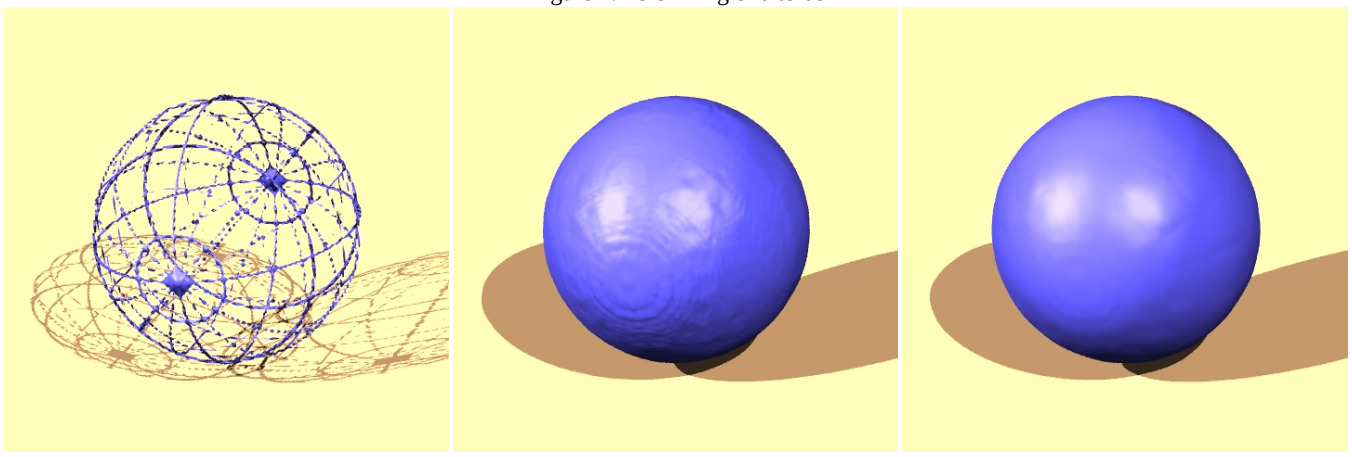
[30] H.K. Zhao, T.F. Chan, B. Merriman, and S.Osher. A variational level set approach to multiphase motion. *J. Comp. Phys.*, 127:179–195, 1996.

[31] H.K. Zhao, S. Osher, B. Merriman, and M. Kang. Implicit and non-parametric shape reconstruction from unorganized points using variational level set method. *Computer Vision and Image Understanding*, 80(3):295–319, 2000.



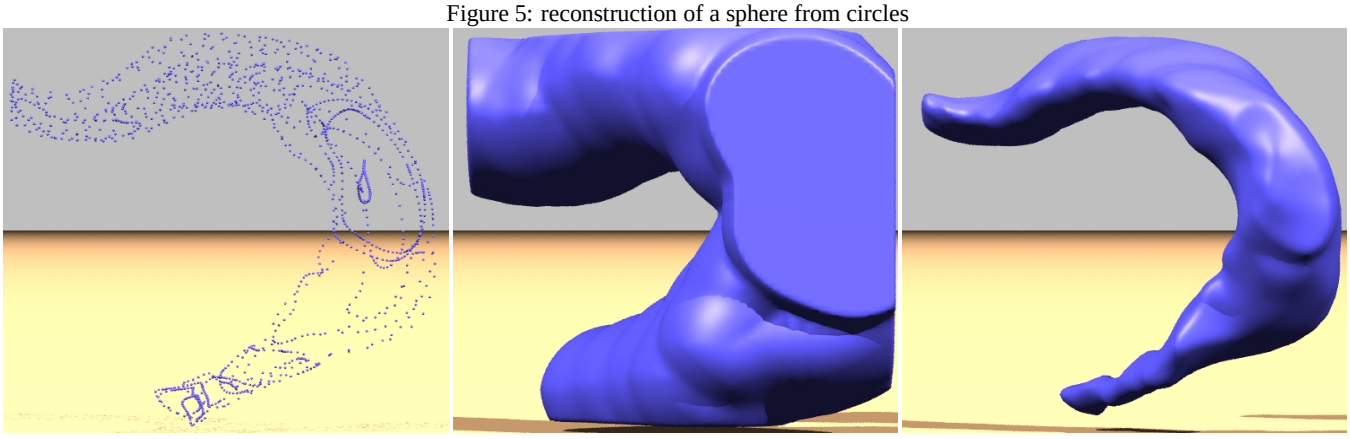
(a) data points (b) front view of the final reconstruction (c) side view of the final reconstruction

Figure 4: hole filling of a torus



(a) initial data (b) reconstruction using convection (c) final reconstruction

Figure 5: reconstruction of a sphere from circles



(a) data points (b) starting surface (c) final reconstruction

Figure 6: reconstruction of a rat brain

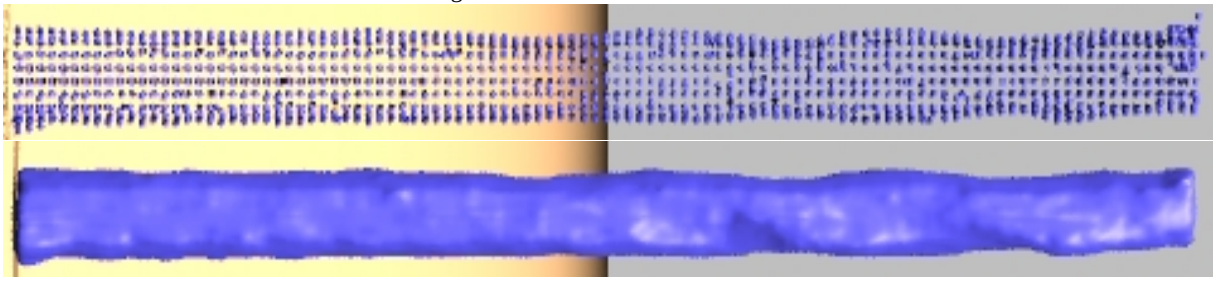
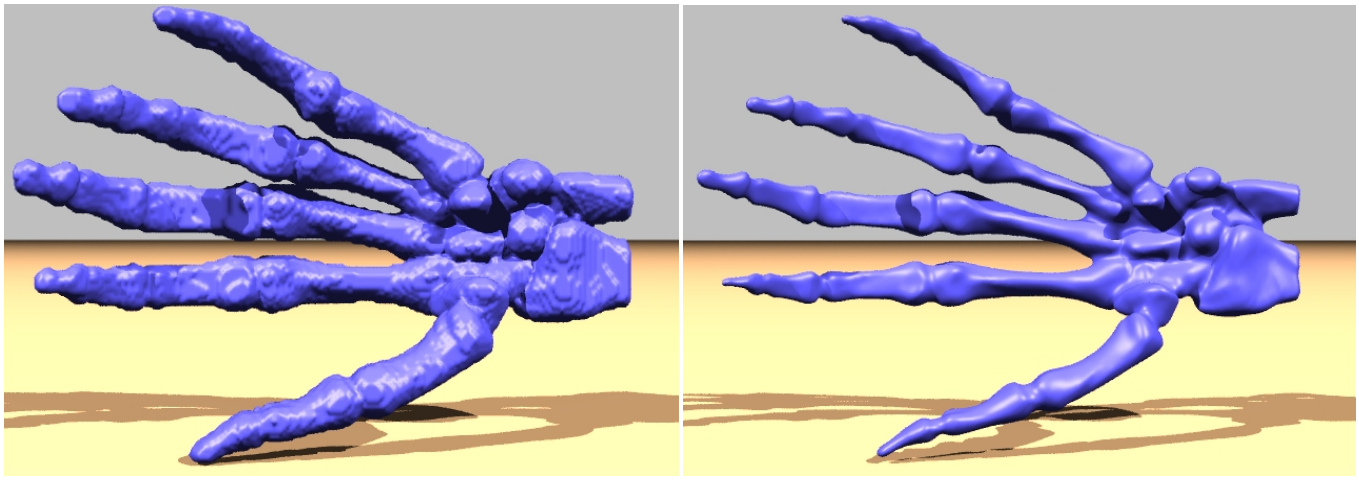
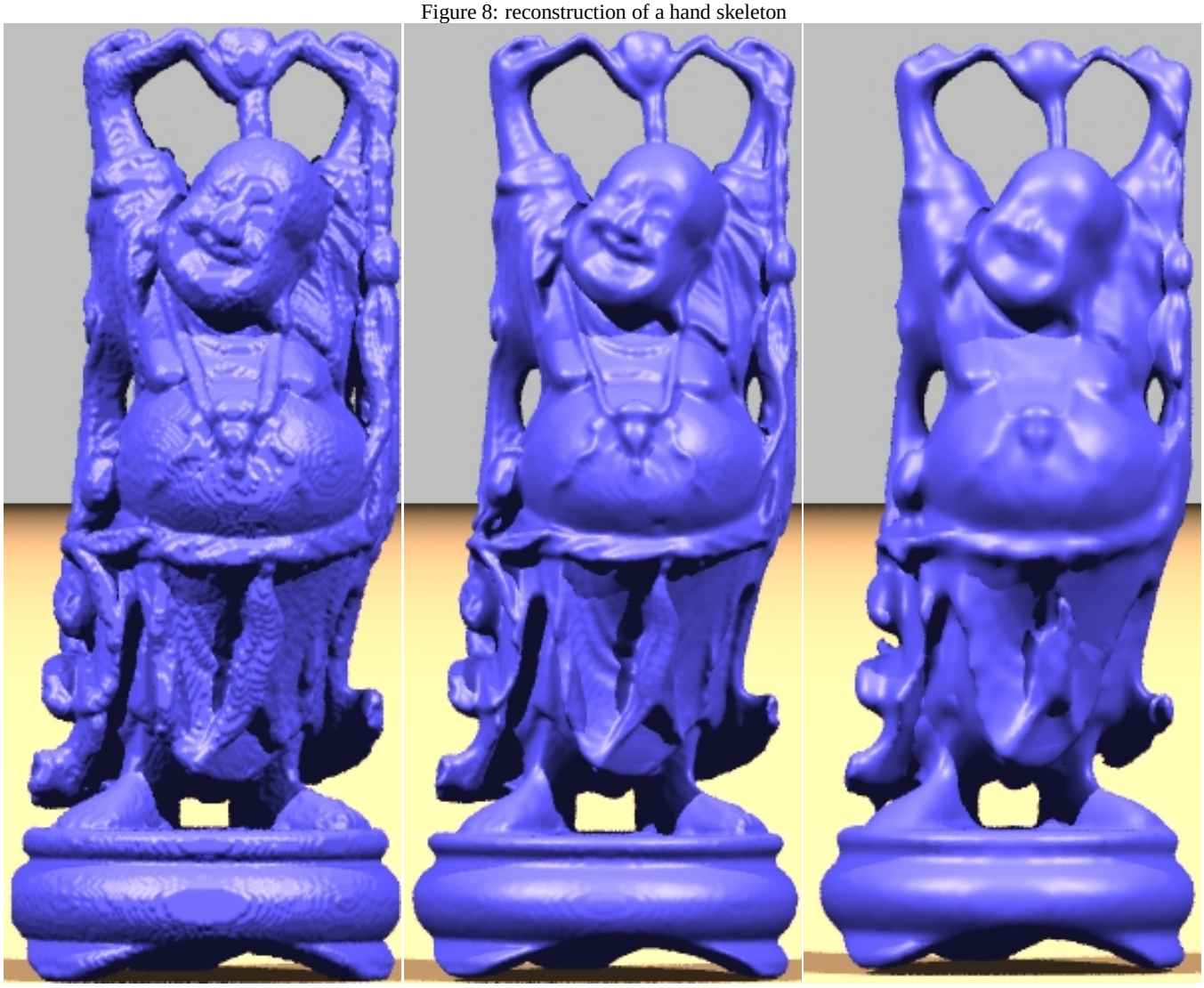


Figure 7: reconstruction of a drill



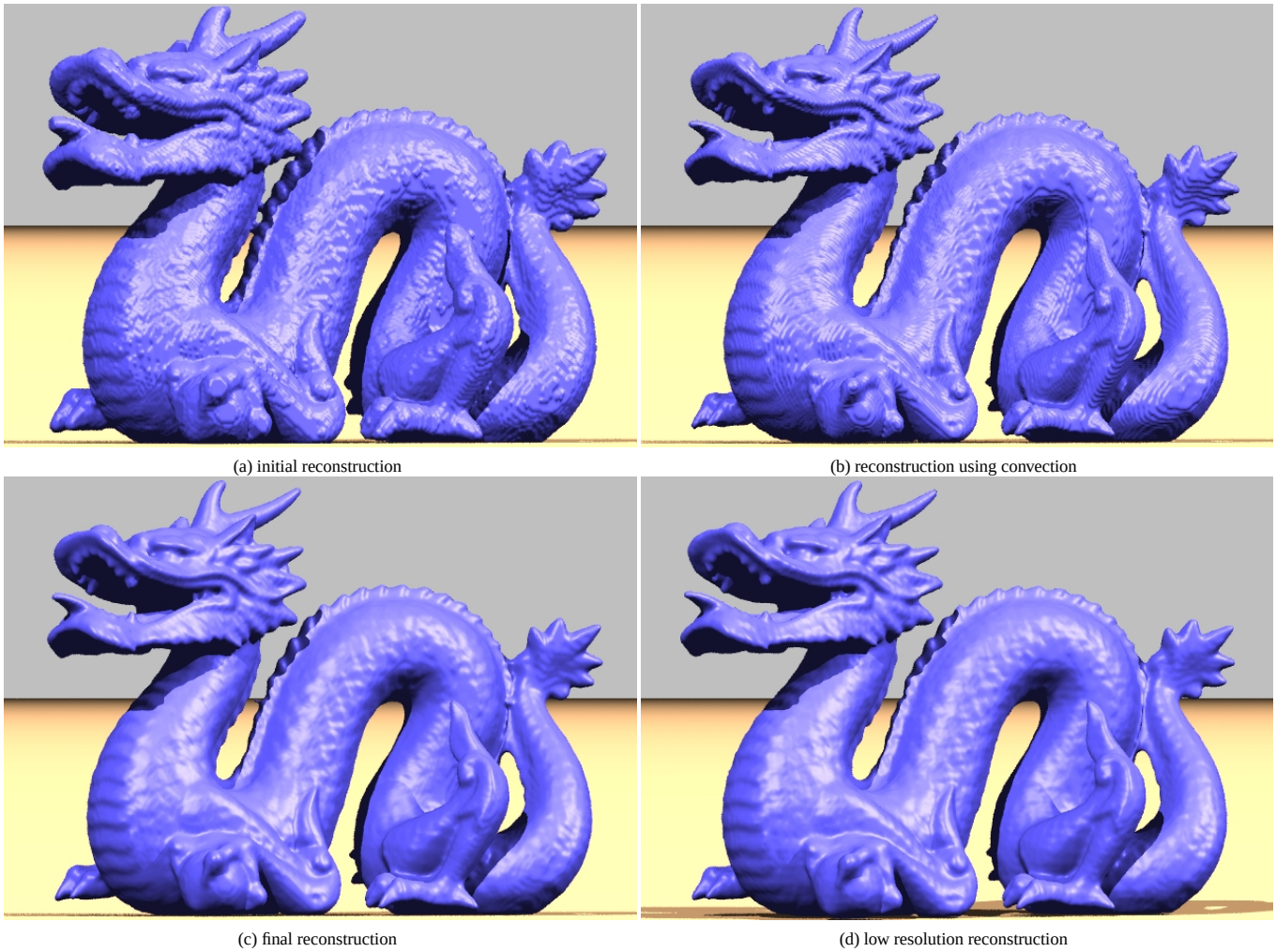
(a) initial reconstruction (b) final reconstruction

Figure 8: reconstruction of a hand skeleton



(a) initial reconstruction (b) final reconstruction (c) reconstruction on a coarse grid

Figure 9: reconstruction of the Happy Buddha



(a) initial reconstruction

(b) reconstruction using convection

(c) final reconstruction

(d) low resolution reconstruction

Figure 10: reconstruction of the dragon

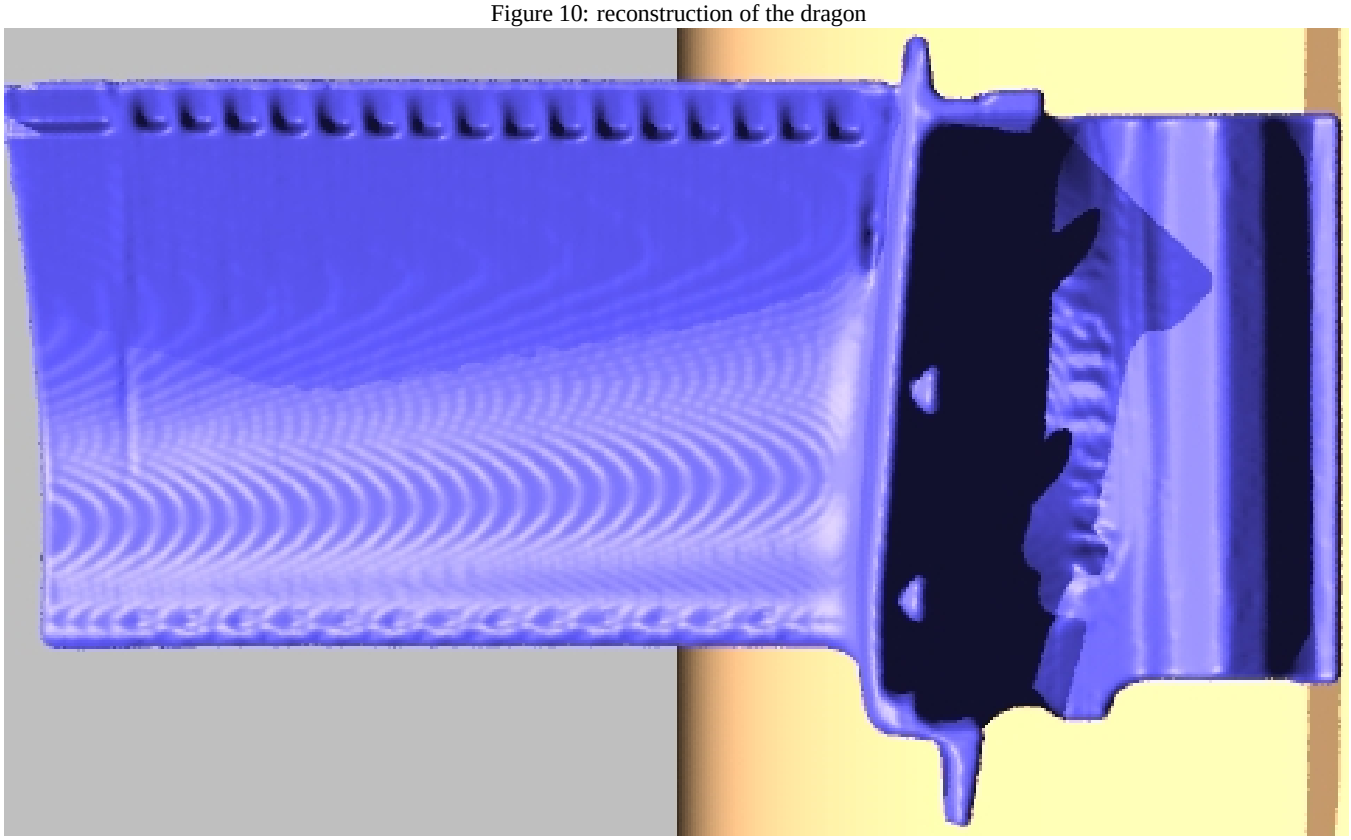


Figure 11: reconstruction of a turbine blade

Dynamic Visibility in an Implicit Framework

Richard Tsai[*†], Li-Tien Cheng[‡], Paul Burchard[§],
Stanley Osher[*¶], and Guillermo Sapiro[||]

February 2002

Abstract

We investigate the problem of determining visible regions in two or three dimensional space given a set of obstacles and a moving vantage point. This is of importance in several fields of study including rendering in computer graphics, etching in materials construction, and navigation. Our approach to this problem is through an implicit framework, where the obstacles are represented by a level set function. An efficient generic multiscale level set method is developed to generate the visible and invisible regions in space. Furthermore, we study the dynamics of shadow boundaries on the surfaces of the obstacles using special level set techniques when the vantage point moves with a given trajectory. In all of these situations, topological changes such as merging and breaking occur in the regions of interest. These are automatically handled by the level set framework here proposed. Finally, we obtain additional useful information through simple operations in the level set framework.

[*]Research supported by ONR N00014-97-1-0027, DARPA/NSF VIP grant NSF DMS 9615854 and ARO DAAG 55-98-1-0323

[†]Department of Mathematics, University of California Los Angeles, Los Angeles, California 90095, email:ytsai@math.ucla.edu

[‡]Department of Mathematics, UCSD, La Jolla, CA 92093-0112. Email: lcheng@math.ucsd.edu

[§]Department of Mathematics, University of California Los Angeles, Los Angeles, California 90095, email:burchard@math.ucla.edu

[¶]Department of Mathematics, University of California Los Angeles, Los Angeles, California 90095, email:sjo@math.ucla.edu

[||]Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, email: guille@ece.umn.edu. Supported by ONR,NSF, PEC ASE, and CAREER.

Contents

1 Introduction

In this paper, we consider the visibility problem described as follows: given a collection of hypersurfaces representing the surfaces of objects, called the occluders, in two or three dimensional space, determine the regions of space or on the surfaces visible to a given observer. In real world applications, this problem must be solved quickly and efficiently, preferably in real time. Generalizations of the visibility problem are just as, if not more, important, such as the case of a moving rather than static observer and the determination of regions visible for all time or invisible for all time in this situation. However, we begin with the basic visibility problem for simplicity, and address parts of the dynamic problem later on. Incidentally, the visibility problem can be reformulated into a problem of determining light and dark regions given a point light source. We occasionally consider this point of view for clarification.

Under this point of view, a more precise set of assumptions we make in the visibility problem includes a space composed of a homogeneous medium and objects with nonreflecting and nondiffracting surfaces. Furthermore, we disregard interference, assuming that the distances between objects are large compared to the wavelength of light. Under these conditions, light rays travel in straight lines and are obliterated upon contact with the surface of an object. Thus a point is called visible with respect to a vantage point, the observer, if the line segment between the point and the vantage point does not intersect any of the obstructing objects or their surfaces in space.

The need for visibility information

Even under these simplifying assumptions, the visibility problem arises as a crucial part of numerous applications in different scientific fields, including rendering, visualization [16], etching [1], the modeling of melting ice [7], surveillance, navigation, and inverse problems, to name a few. In the case of computer graphics and rendering, for example, determination of the visible portions of object surfaces allows for those portions alone to be rendered, thus saving a lot of costly computation. Additionally, there are recent variational formulations for surface reconstructions that require the solution of the visibility problem [17].

While explicit surfaces, for example triangulated surfaces, are used in a majority of computer graphics and vision applications, implicitly represented surfaces are gaining more attention. The advantages can be seen in the automatic resolution of surfaces as well as the incorporation of geometric information and the handling of various surface topologies afforded, for example, by a level set framework (see, e.g., [6, 9, 15, 16, 17, 26]). Currently there are numerous algorithms for solving

the visibility problem using explicit surface representations. For example, the work of [12] and [14] uses linearity to process triangulated surfaces. A detailed review of related work on the visibility problem, especially concerning explicit surfaces, can be found in [13]. Furthermore, there are a variety of visibility algorithms from computational geometry (see, e.g., [2, 3]). Visibility algorithms for implicit surfaces mostly consist of sending rays out from the point source and testing for intersections with the surfaces of the objects using information arising from the implicit formulation. Our proposed method for ray tracing is different. In essence, we send out rays in an implicit manner so as to propagate the causality relation of visibility. We describe this approach in more detail below.

Level set approach

We propose to solve the static and dynamic visibility problems in an implicit manner using a level set formulation. The level set method was first proposed by Osher and Sethian [20] as a PDE (Partial Differential Equation) based numerical device to capture moving interfaces. Over the years, a level set calculus has been developed that allows for the application of the level set method to a multitude of problems and situations. See, for example, the review paper [19] for an overview on the basics as well as recent advances in level set methods. We only recapitulate here that implicit PDE approaches such as the level set approach retain the important self interpolating property when propagating interfaces, thus obtaining good resolution of the interfaces. Furthermore, Boolean operations on sets, including finding curves of intersections and the trimming commonly required in CAD (Computer Aided Design) are easy to implement in a level set framework.

In our case of visibility, a real valued two or three dimensional function ϕ, called the level set function, is introduced. The zero level set of this function represents the surfaces of the occluding objects. Furthermore, we require that the points where ϕ is negative represent the interior of the objects. Several algorithms have been developed in the literature to efficiently obtain this representation. A level set method for visibility will use this function ϕ whenever the objects are considered.

One idea in determining whether a point is visible to a given observer is to compare the geodesic and Euclidean distances between the observer and that point. See [24] for an example of this approach. The geodesic distance between two points is the distance in the space in the presence of obstacles, namely the objects. Let $\mathbf{x}$ represent the point of interest and $\mathbf{x_o}$ represent the observer point. The geodesic distance can thus be calculated by solving the Eikonal equation

$$H(\phi)|\nabla u| = 1,$$

where H is the one dimensional Heaviside function, with condition $u(\mathbf{x_o}) = 0$. Thus the point $\mathbf{x}$ is occluded if and only if

$$u(\mathbf{x}) > |\mathbf{x} - \mathbf{x_o}|.$$

However, this algorithm is at best $O(N \log N)$, where N is the number of grid points (see, e.g., [25]). This may not be optimal, making it too slow for applications requiring real time computations. Furthermore, numerical implementation of the Heaviside function may cause problems for accuracy.

Our level set approach

We present here a few level set based algorithms for determining various types of visibility information in any dimension, though three dimensions is probably the one of interest. We first introduce a multiresolution algorithm for solving the visibility problem for a given fixed vantage point. This algorithm constructs the occlusion boundary, the interface separating visible from invisible. At each resolution level, we solve a radially defined causality relation on a given grid in one pass, obtaining not only a conservative estimate of the visible and invisible regions but a locally second order approximation of the occlusion boundary. In addition, our algorithm is independent of both the convexity of the occluders and the grid geometry, and its parallelization is straightforward.

The level set framework is especially important as it handles occluder fusion, where the occlusion boundary merges during the construction process. Furthermore, the implicit representation, though not as effective on occluders which are open surfaces, can still handle this case by considering them as very thin hypersurfaces.

Dynamic visibility

In the second part of the paper, we extend our study to the dynamic visibility problem. In this case, we consider a moving vantage point. Obviously the static visibility problem can be applied at each time to solve this problem, and our algorithm can be used to solve it efficiently enough. However, this static approach does not give us other useful information about the dynamics; for instace, how fast a point in space will become visible or invisible. In many cases, the problem can be solved even faster if the visibility at a previous time is used effectively to produce visibility at future time.

Thus we study the dynamics of curves on the occluders that separate light and dark regions on the occluders. The curves in fact can be represented using a level set approach, following the work of [5, 8, 10]. We also study other types of curves,

called horizons, and their motions which form a superset of the curves separating light from dark but can be constructed quickly. We rigorously derive motion laws for all these types of curves and evolve them under the level set framework. This framework allows for topological changes which may occur in the curves and its self interpolating property automatically produces well resolved results. Finally, we derive an emergence-time estimate to predict an occluded object's emergence into view. Thus our work complements the book of Cipolla and Giblin [11] which discusses the reconstruction of shape from the perspective (orthogonal) projection of the horizons. This is exactly what we are trying to evolve.

Notations

Through out this paper, we use the following notation:

- The space in which we work will be $\mathbb{R}^d$, where $d = 2$ or 3.
- $\mathbf{x_o}$ denotes the position of the vantage point, or observer. We further assume that $\mathbf{x_o}$ never lies in the interior of the objects.
- Ω is a set of connected domains whose closure denotes the objects in question. Furthermore, let $\Gamma = \partial\Omega$.
- ϕ denotes the level set function representing the objects of interest. We may further assume that ϕ is the signed distance function to Γ. This particular level set function can be efficiently computed using fast algorithms such as the fast marching method of [25] or fast sweeping methods.
- We define the view direction vector pointing from $\mathbf{x_o}$ to $\mathbf{x}$ by $\nu(\mathbf{x_o}, \mathbf{x}) = (\mathbf{x} - \mathbf{x_o})/|\mathbf{x} - \mathbf{x_o}|$. When the context is clear, we will drop the arguments and write simply $\nu(\mathbf{x})$ or ν.
- Let $\mathbf{x_1}$ and $\mathbf{x_2}$ denote two points in space. We say $\mathbf{x_1} \preceq \mathbf{x_2}$ ($\mathbf{x_1}$ is "before" $\mathbf{x_2}$) if the conditions $\nu(\mathbf{x_o}, \mathbf{x_1}) = \nu(\mathbf{x_o}, \mathbf{x_2})$ and $|\mathbf{x_1} - \mathbf{x_o}| \leq |\mathbf{x_2} - \mathbf{x_o}|$ are satisfied. We also define the strict relation $\prec$ if the condition $|\mathbf{x_1} - \mathbf{x_o}| \leq |\mathbf{x_2} - \mathbf{x_o}|$ above is replaced by $|\mathbf{x_1} - \mathbf{x_o}| < |\mathbf{x_2} - \mathbf{x_o}|$.
- A point $\mathbf{y} \in \Gamma$ is called a horizon point if and only if $\nu(\mathbf{x_o}, \mathbf{y}) \cdot \mathbf{n}(\mathbf{y}) = 0$, where $\mathbf{n}(\mathbf{y})$ is the outer normal of Γ at $\mathbf{y}$. The horizon thus refers to the set of horizon points.
- A point $\mathbf{y} \in \Gamma$ is a cast horizon point if and only if there is a point $\mathbf{y}^*$ such that: 1) $\mathbf{y}^* \prec \mathbf{y}$ and 2) $\mathbf{y}^*$ is a horizon point. The cast horizon thus refers to the set of cast horizon points.

- The visible contour refers to the set of visible points of the horizons and cast horizons.

2 Implicit ray tracing

We now set up the foundation of our approach and derive properties of ray tracing of a single point source in an implicit framework. The motivation is as follows: we observe that the visibility status of points sharing the same radial direction centered at the vantage point satisfy a causality condition. This means if a point is occluded, then all other points farther away from the vantage point in the same radial direction are also occluded, i.e., if $\mathbf{x_1}$ is occluded and $\mathbf{x_1} \preceq \mathbf{x_2}$, then $\mathbf{x_2}$ is also occluded.

This fact can be described more rigorously as follows. Define

$$\rho(\mathbf{p}) = \begin{cases} \min_{\mathbf{x}\in\mathbb{R}^d}\{|\mathbf{x}-\mathbf{x_o}| : \nu(\mathbf{x_o},\mathbf{x}) = \mathbf{p}, \phi(\mathbf{x}) \le 0\} & \text{if exists} \\ \infty & \text{otherwise,} \end{cases} \tag{1}$$

giving the distance between $\mathbf{x_o}$ and the closest point on Γ in the direction of $\mathbf{p}$ from $\mathbf{x_o}$. Thus a given point $\mathbf{x}$ is invisible if $\rho(\nu(\mathbf{x},\mathbf{x_o})) \le |\mathbf{x}-\mathbf{x_o}|$. Refer to Figure 1 for an example and clarification. Therefore, we can define the visibility indicator

$$\Xi(\mathbf{x},\mathbf{x_o}) := \rho(\nu(\mathbf{x},\mathbf{x_o})) - |\mathbf{x}-\mathbf{x_o}|,$$

so that $\{\Xi \ge 0\}$ is the set of visible regions in $\mathbb{R}^d$ and $\{\Xi < 0\}$ is the set of occluded regions.

From another view point, the problem becomes: compute

$$\psi(\mathbf{x}) := \min_{\xi\in L(\mathbf{x_o},\mathbf{x})} \phi(\xi),$$

where $L(\mathbf{x_o},\mathbf{x})$ is the line segment connecting $\mathbf{x_o}$ and $\mathbf{x}$. Thus if $\psi(\mathbf{x})$ is negative, then $\mathbf{x}$ is occluded. In fact, our implicit ray tracing algorithm is is an approximation of this formula.

Implicit formulation

Our implicit framework encodes visibility information in a Lipschitz continuous function ψ so that a point $\mathbf{y}$ is visible if $\psi(\mathbf{y}) \ge 0$ and invisible if $\psi(\mathbf{y}) < 0$. Thus we can compute the value of $\psi(\mathbf{x})$ by

$$\psi(\mathbf{x}) = \min(\psi(\mathbf{x}'), \phi(\mathbf{x})) \tag{2}$$

where $\mathbf{x}'$ is some point "right before" $\mathbf{x}$ in the ray direction. We can therefore start from the vantage point $\mathbf{x_o}$ and update the grid points following the ray directions outwards. A simple algorithm for this reads:

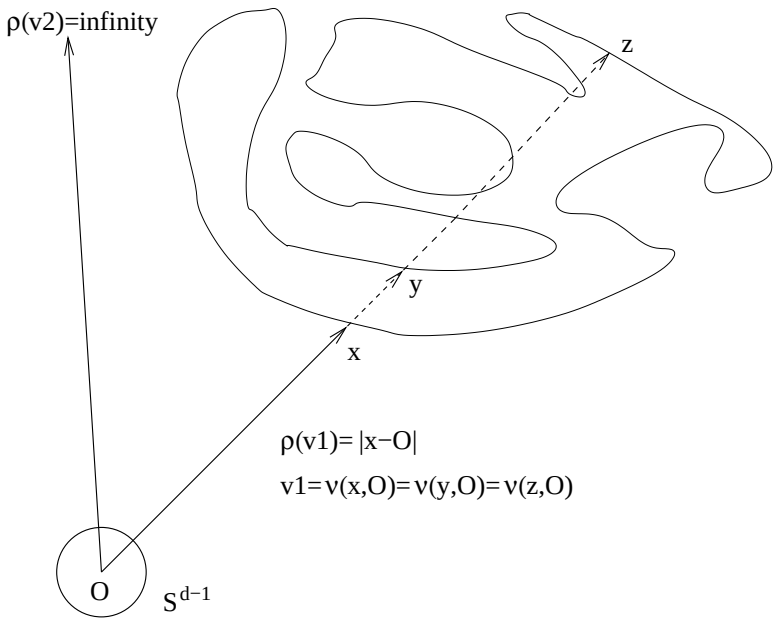


Figure 1: Illustration of ρ

1. Set $\psi(\mathbf{x_o}) = \phi(\mathbf{x_o})$.
2. Do a star-shaped[1] updating sequence on the grid.
3. For each grid point $\mathbf{x}$, choose $\mathbf{x}'$ depending on the grid geometry.
4. Compute the value of $\psi(\mathbf{x})$ via (2).

Each grid node is visited in a specified order that maintains the causality. As long as the updated grid nodes form a star-shaped region centered at the vantage point, causality is maintained. See the Appendix for an example of such an updating method. Due to the minimization and the linear interpolation (see Section 5.1) used to find $\mathbf{x}'$, the algorithm is l_∞-stable. More precisely, we have

$$\phi_{\text{int}}(\mathbf{x}') \leq \max\{|\phi(x_l)| : x_l \text{ are the points used in the interpolation}\},$$

where ϕ_{int} is the interpolant we constructed. Please see Section 5.1. Figure 2 shows what ψ should look like in a one space dimension setting.

[1] See 5.2

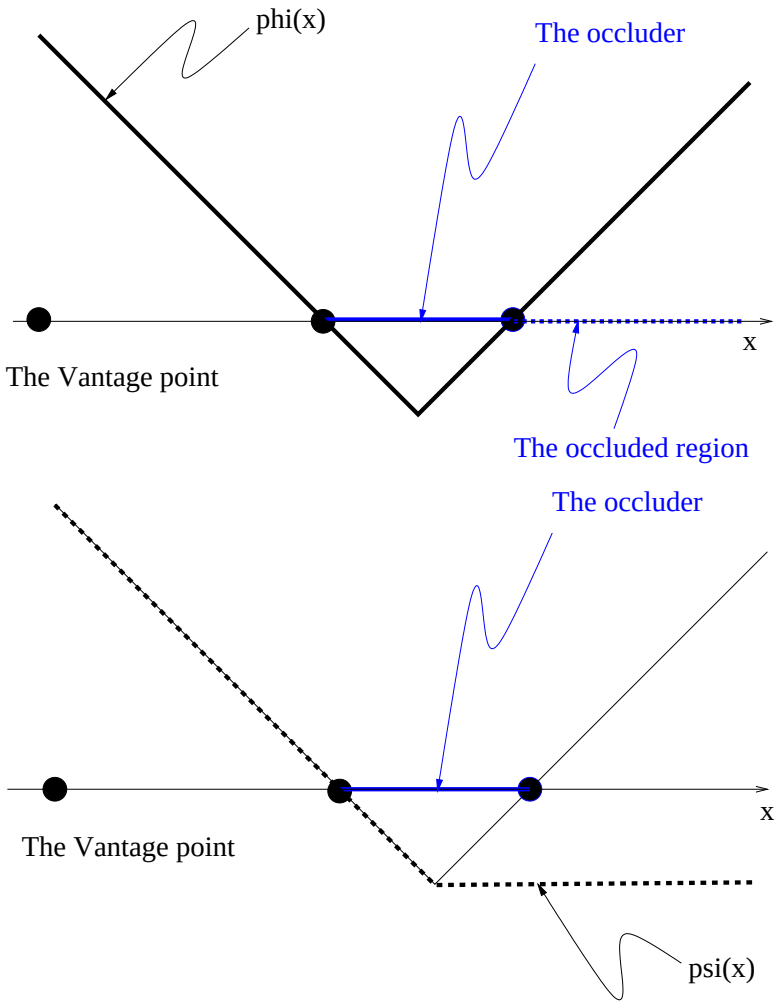


Figure 2: A demonstration of the motivation of our implicit ray tracing algorithm in one dimension.

Volumetric visibility processing

If we want to determine volumetric visibility information, i.e., we want to find regions that are occluded from a given rectangular region (view cell), the above algorithm can be modified for this purpose. For simplicity, assume that the view cell degenerates to a line with end points $\mathbf{x}_1$ and $\mathbf{x}_2$. At each point $\mathbf{x}$, we compute $\mathbf{x}'_1$ and $\mathbf{x}'_2$ as in step 2 with respect to $\mathbf{x}_1$ and $\mathbf{x}_2$. The update formula then becomes: $\psi(\mathbf{x}) = \min(\phi(\mathbf{x}), \max(\psi(\mathbf{x}'_1), \psi(\mathbf{x}'_2)))$.

2.1 Multi-resolution calculation

Finding inside/outside

Any multi-resolution approach of the visibility problem requires the skipping of large regions which we know a priori are either visible or invisible. This hinges upon the ability to determine whether any given voxel is completely "inside" or "outside" of the objects. This can be done conservatively with the help of the Lipschitz constant of the embedding level set function.

Let C be the Lipschitz constant of ϕ. Let $\mathbf{x}_c$ be the center point and $\mathbf{x}_i$ the vertices of the given voxel V. If

$$\phi(\mathbf{x}_i) + C|\mathbf{x}_c - \mathbf{x}_i| < 0 \ \ \forall\, i, \tag{3}$$

then we know $\phi|_V < 0$ ($V \subset \{\phi < 0\}$). Conversely, if

$$\phi(\mathbf{x}_i) + C|\mathbf{x}_c - \mathbf{x}_i| > 0 \ \ \forall\, i, \tag{4}$$

then we know $\phi|_V > 0$. Since our embedding function is the signed distance function, the Lipschitz constant $C = 1$.

Correspondingly, we also have:

$$\phi|_V < 0 \ \text{ if } \ \phi(\mathbf{x}_c) + C|\mathbf{x}_i - \mathbf{x}_c| < 0 \ \ \forall\, i,$$

and

$$\phi|_V > 0 \ \text{ if } \ \phi(\mathbf{x}_c) + C|\mathbf{x}_i - \mathbf{x}_c| > 0 \ \ \forall\, i.$$

A voxel V is occluded if it lies completely inside the an object or if it is "behind" an occluded voxel $\tilde{V}$.This idea can be implemented by a careful reinterpretation of formula (2). A similar condition for determining completely visible voxels can also be easily derived.

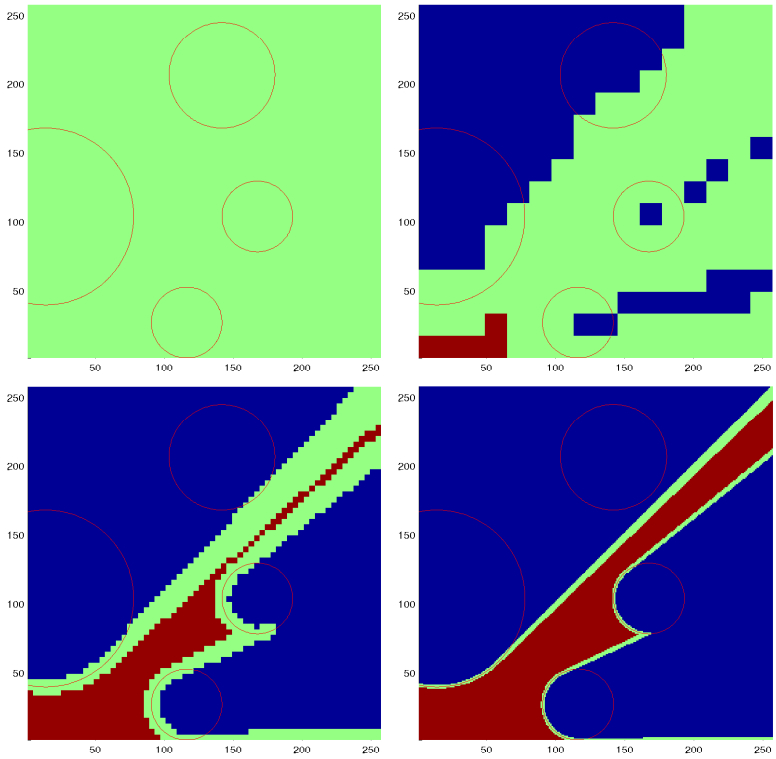


Figure 3: This is a schematic diagram for the multi-resolution algorithm. Occluded voxels are depicted in blue and visible ones in red. The regions are target for next level refinement.to be refine. The red curves represent the boundaries of the occluders, and the vantage point is positioned at $(1, 1)$. The sizes of the voxels are: 64×64, 16×16,4×4, and 1×1.

2.2 A Multi-resolution algorithm

Let h give the resolution of our grid such that smaller h means high resolution (i.e. finer grid). In practice, we can simply set h to be the mesh size. Under resolution h, we use (i^h, j^h, k^h) to denote the grid indices and V^h for a voxel in the grid. We will drop the superscripts of these indices when the context is clear.

Algorithm: (Multi-resolution visibility sweeping)

Let $\{h_l : h_l > h_{l+1}, l = 0, \cdots, m\}$ be the set of resolutions of interest. For each resolution level h, decending from h_0 to h_m,

1. Compute ψ on the (i^h, j^h, k^h) which do not lie on a voxel marked either visible or occluded.

2. For each voxel V^h which does not lie on a voxel marked either visible or occluded, mark V^h to be visible or occluded according to formulas (3) and (4).

We remark that the Lipschitz constant of the linearly interpolated ψ is can be taken from that of ϕ. Therefore, Step 2 above is well defined. Please see Figure 3 for a demonstration of this algorithm.

As for complexity, the multiresolution algorithm for constructing visiblity information should be an $O(N^{d-1} \log N)$ algorithm in terms of speed. Here $N = 1/h$ where h is the smallest spatial stepsize used in the multiresolution framework and d is the dimension of the space. The N^{d-1} part of the complexity comes from the fact that a codimension one hypersurface in d dimensional space is being generated under fast sweeping and the $\log N$ part comes from multiresolution. The memory allocation of our algorithm is also $O(N^{d-1} \log N)$, with the $\log N$ part once again due to multiresolution. In practice, our algorithms have proven to be very fast, obtaining detailed visiblity information in almost real-time. For example, it takes less than a second to perform this algorithm running on only one resolution (meaning no multi-resolution), on a grid with 100^3 cells, on a moderate PC.

2.3 Multi-scale considerations

If we consider the visibility problem in applications related to human vision, such as 3D virtual environment rendering, it is natural to put a scale parameter into the size of the objects related to the distance of the object from the vantage point. We want to ignore certain **isolated and small** objects that are **far away** from the vantage point using this information. It is important to notice that a collection of closely positioned small objects can form a visible ensemble, seen for example in clouds and trees.

Using the level set representation of the virtual environment in conjunction with PDEs, we are able to deal with this issue easily without explicitly considering each object separately. The idea is to *dilate* the interface first so that small objects can merge to form ensembles of larger size. We then *shrink* the interfaces (one possibility is to perform curvature driven motion) such that remaining small objects will disappear. This approach follows the regularization effect of viscosity solution theory for Hamilton-Jacobi Equations. It is basic mathematical morphology, and can be done easily, see e.g. [4][23].

3 Dynamic visibility

We now consider the case in which the vantage point is moving. Naturally visibility information changes according to the position of the vantage point. We are interested in how visibility changes and when hidden objects become visible. This amounts to studying how the boundaries between visible and invisible regions move with respect to the vantage point motion. We first remark that these boundaries are hypersurfaces in regions outside of the occluders, and that the Gaussian curvature on such surfaces is 0.

For a single convex object, the horizon determines the visibility information on the surface. Therefore, tracking the motion of the horizon for all time gives us incremental information on the change of the visible portion of the object.

We formulate the visibility problem so that the points which are on the boundaries of the visible regions on the surfaces can easily be identified. The dynamics of these points are derived so that one can track the visible regions according to the motion of the vantage point $\mathbf{x_o}$.

The points forming the boundaries of visible regions on given surfaces can be placed into two categories:

- points that are part of the horizon;
- points that border *shadows cast by some surface* (cast horizon).

Thus, two types of motions need to be investigated, namely, that of the horizon and of the cast horizon. The motion of the horizon is characterized by the orthogonality constraint and it, in turn, becomes a part of the constraints of the cast horizon motion.

In a level set formulation, we want to create a level set function whose zero level set captures the points described above. We need a description relating each point on the cast horizon to a point on the horizon of the surface casting the shadow.

Furthermore, our description should be global, that is, quantities should vary "continuously" with respect to points not on the surface. This requirement is essential for the success of our level set formulation.

Assuming that there is no singularity in the velocity field of the horizon or cast horizon motion, and there are no other considerations, the level set approach of tracking the visible contours is optimal. With the fast sweeping algorithms and local storage strategies, the complexity of the level set approach to track the horizon and cast horizon curves is formally $\mathcal{O}(N)$ in operation counts and in storage. Here, the number N is the number of points used to resolve the curves. In actual applications, there are other aspects that affects the overall complexity of this approach. We will address this point in a later subsection.

3.1 Finding the horizon and the cast horizon implicitly

Horizons and cast horizons are objects of codimension 2. We may therefore categorize these objects by the intersection of the zeros of two level set functions. Furthermore, for numerical reasons, we want the two zero level sets to be more or less orthogonal to each other near their intersection.

Finding the horizon

We extend the orthogonality condition that defines the horizon and arrive at

$$\hbar(\mathbf{x}, t) = (\mathbf{x} - \mathbf{x}_0) \cdot \nabla\phi(\mathbf{x}). \tag{5}$$

$\hbar$ determines the visibility of any convex object embedded in ϕ :

$$\{\hbar(\mathbf{x}) \leq 0\} \bigcap \{\phi = 0\} \iff \text{visible}.$$

In general cases, where there are multiple objects (convex and nonconvex), $\hbar$ does not give exact visibility information anymore. It just provides local visibility information just as local extrema may not be absolute extrema. Thus clearly, for objects hiding completely behind other objects, $\hbar$ will still be non-negative on the parts of the surface facing the source. Instead, $\hbar$ gives a conservative "estimate" of the shadow:

$$\{\hbar(\mathbf{x}) > 0\} \bigcap \{\phi = 0\} \implies \text{invisible}.$$

Thus the visible horizon is a subset of $\{\phi = 0\} \bigcap \{\hbar = 0\} \bigcap \{\psi \geq 0\}$, where ψ is the visibility function coming from our static algorithm. Figure 4 gives an example of horizons found this way.

Finding the cast horizon

How do we find the cast horizon? The idea is to overshoot the shadow boundaries generated by the visible horizon when it hits another part of Γ, creating a level set piece $\tilde{\psi}$ in that neighborhood, and then propagate ψ as usual. $\{\tilde{\psi} = 0\}$ will cut through Γ on the cast horizon, therefore providing an implicit representation of it. We can even make $\{\tilde{\psi} = 0\}$ perpendicular to Γ locally around the intersection by iterating on the following PDE used in [18]:

$$\tilde{\psi}_\tau + \text{sgn } (\phi)\nabla\tilde{\psi} \cdot \frac{\nabla\phi}{|\nabla\phi|} = 0.$$

A more direct approach is to define $\tilde{\psi}$ on a grid point to be the "upwind" value of ϕ. This will introduce an overshoot of the size of a mesh size. Alternatively, we notice that the occlusion generated by the set $\{\hbar \geq 0\}\bigcap\{\phi \leq 0\}$ is the same[2] as $\{\phi \leq 0\}$. Therefore, we can define $\tilde{\psi}$ from the result of our algorithm under the configuration $\{\hbar \geq 0\}\bigcap\{\phi \leq 0\}$. Figure 4 shows the operations described above in a simple two circle setting.

With these characterizations and the visibility result, we can easily identify the visible contours. See Figures 5, 6, and Figures 7, 8[3] for examples. In these figures, the visible portions of the horizons and the cast horizons are depicted as cyan and yellow curves respectively. A green circle is drawn to reveal the location of the vantage point in each setting. The boundaries between visible and invisible regions are represented by blue surfaces. We observe that the blue surfaces cut through the objects exactly at the visible contours.

3.2 The dynamics of the horizon

Let $\mathbf{x}_o(t)$ be the position of the vantage point and $\mathbf{x}(t)$ be a corresponding point on the horizon at time t. We first consider a single convex occluder Ω embedded by the signed distance function ϕ. Let $n(x)$ denote the outer normal of $\partial\Omega$ at $\mathbf{x}$. This translates into the following constraints on $\mathbf{x}(t)$:

$$\begin{cases} \phi(\mathbf{x}(t)) = 0 \\ (\mathbf{x} - \mathbf{x_o}) \cdot \nabla\phi(\mathbf{x}) = 0 \end{cases} \tag{6}$$

In two dimensions, we can invert the above constraints and derive that

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \frac{1}{\kappa}\frac{\dot{\mathbf{x}}_\mathbf{o} \cdot n(\mathbf{x})}{|\mathbf{x} - \mathbf{x}_o|}\nu(\mathbf{x}). \tag{7}$$

[2] modulo a small subset of $\{\phi \leq 0\}$, which we know is invisible by definition.

[3] The terrain data is obtained from `ftp://ftp.research.microsoft.com/users/hhoppe/data/gcanyon/.`

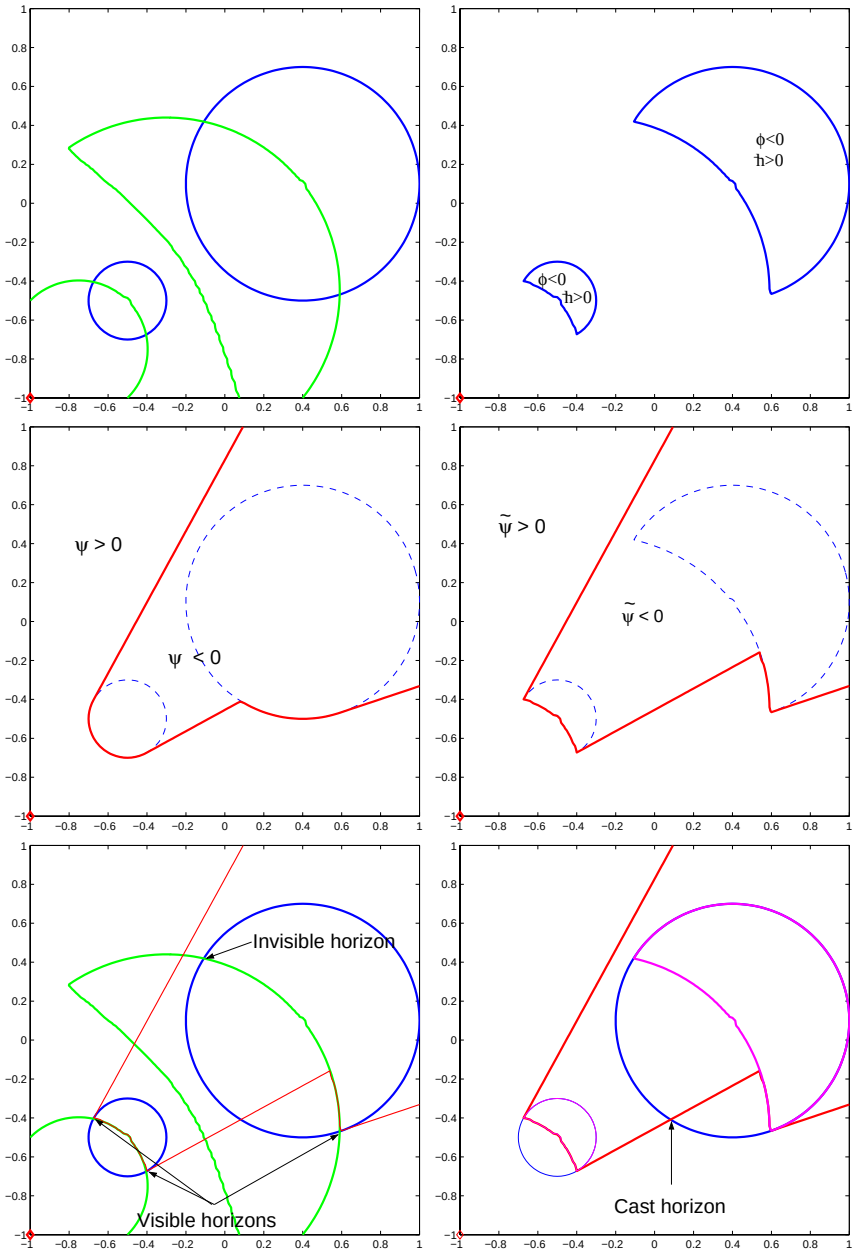


Figure 4: Finding the visible horizons and their casts. The occluders are the two circles depicted by the blue curves, and the vantage point is located at $(-1,-1)$. The green curves are the zero level set of $\hbar$. Visible horizons and their casts are characterized by the intersections of different level set functions as described in the text.

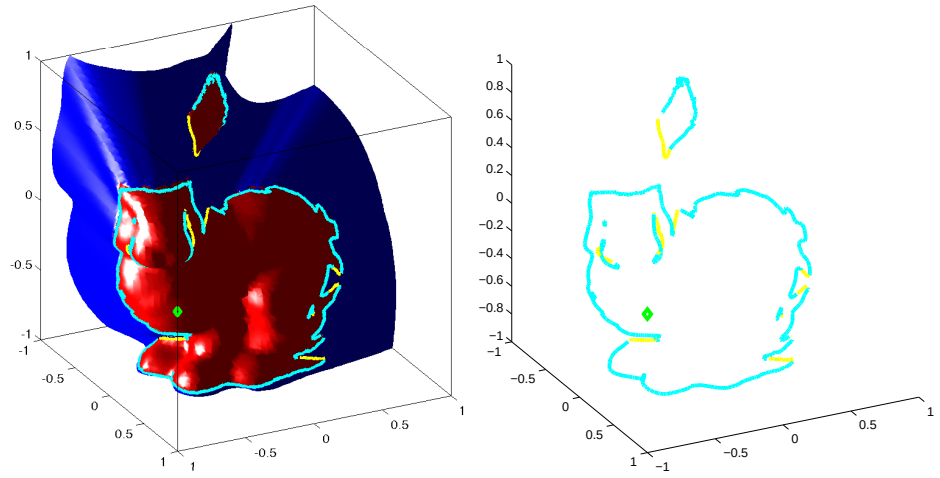


Figure 5: Visible contour (portions of horizon and cast horizon that are visible)

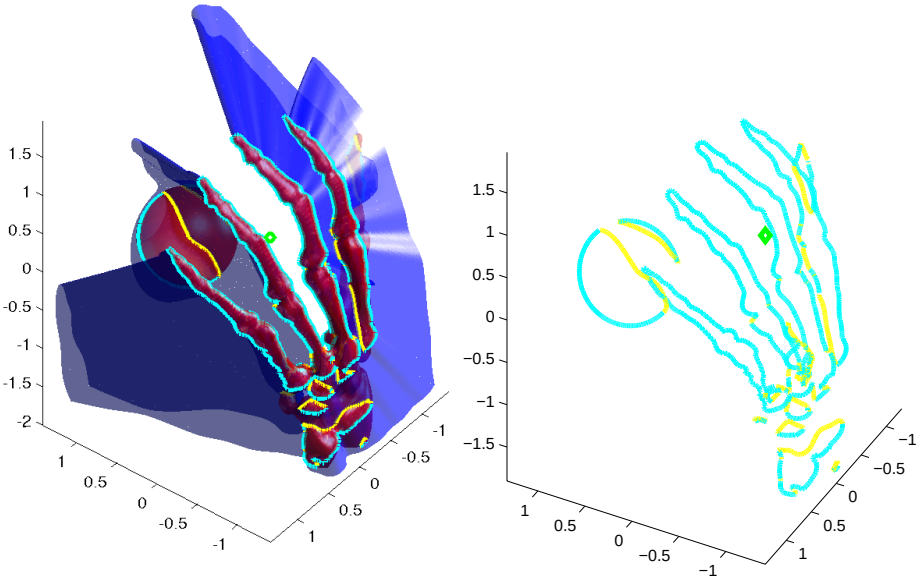


Figure 6: Visible contour (portions of horizon and cast horizon that are visible)

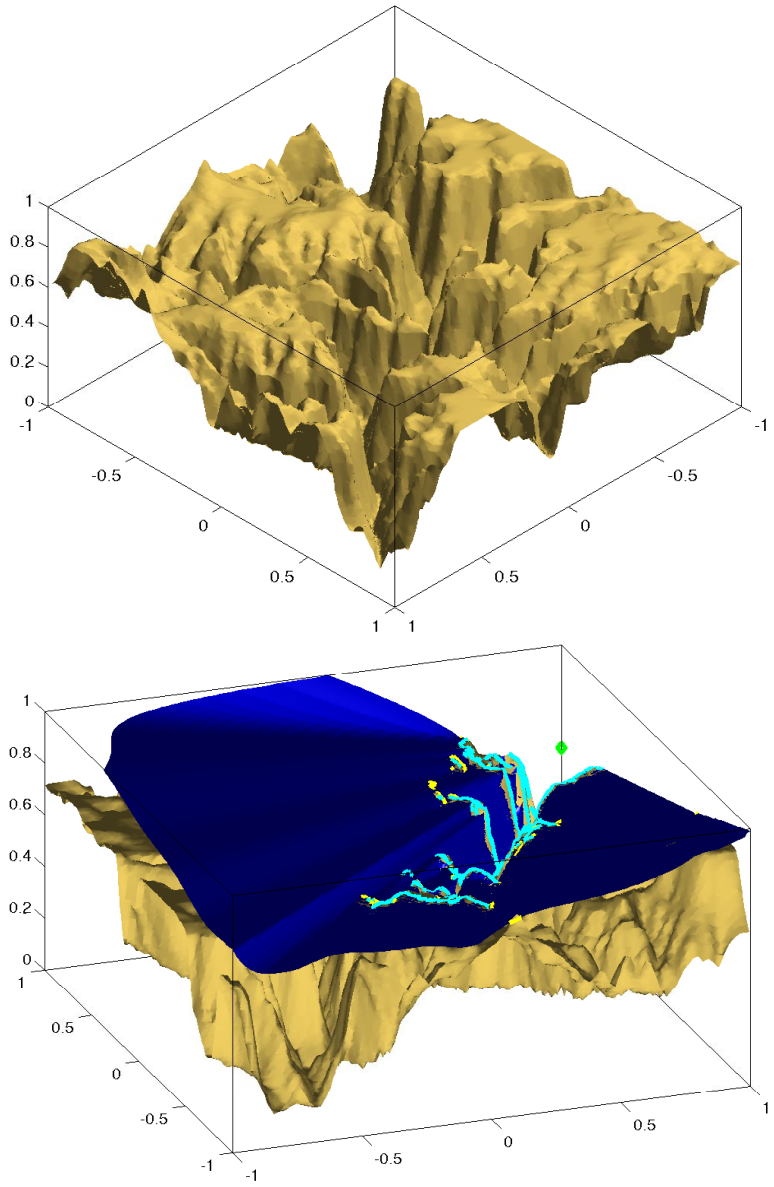


Figure 7: Horizons and cast horizons obtained from the elevation data of Grand Canyon.

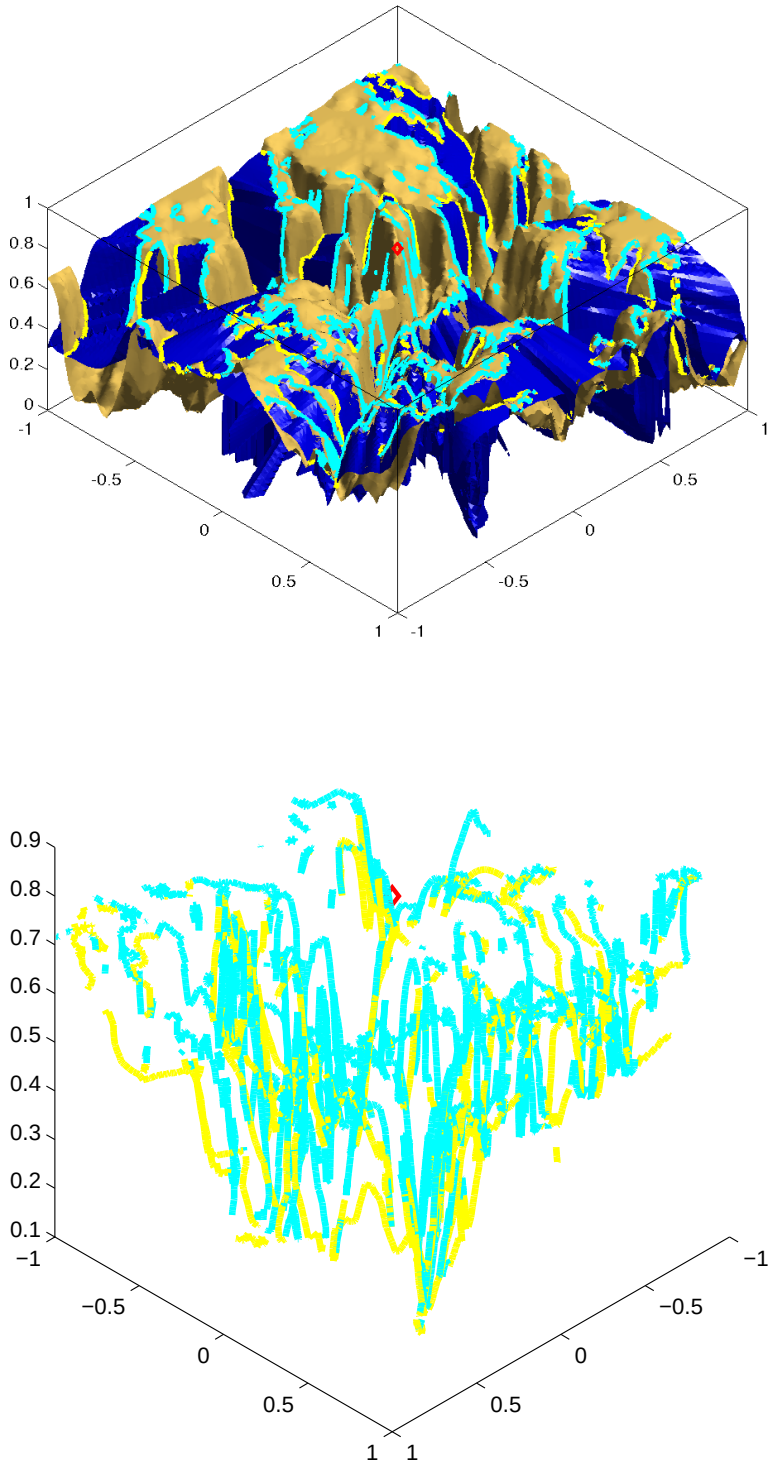


Figure 8: Horizons and cast horizons obtained from the elevation data of Grand Canyon.

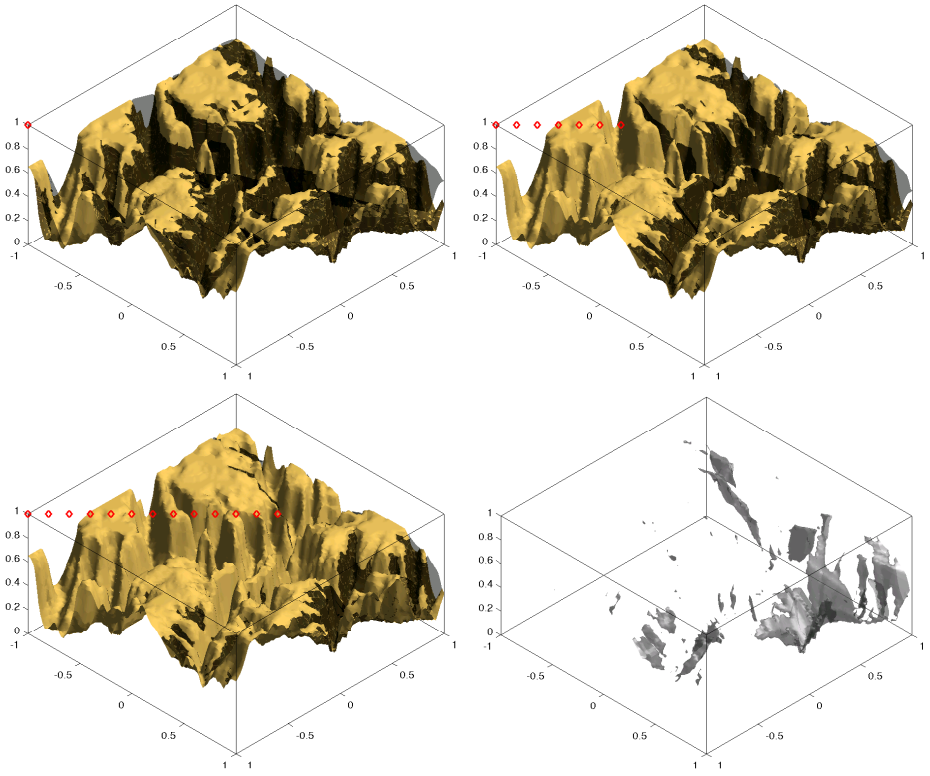


Figure 9: By taking the intersection of the occlusion during a trajectory of the observer, we can find the cumulative occlusion easily and efficiently. The following pictures show a progression of the cumulative occlusion subject to an observer (“spy plane”) moving across a region of Grand Canyon.

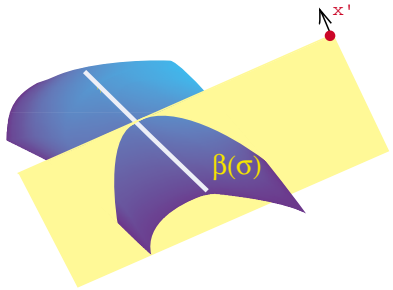


Figure 10:

Here, κ is the curvature of the occluding surface at $\mathbf{x}$. In three space dimensions, the horizon becomes a closed curve $\Gamma(s) = \mathbf{x}(s,t)$, where s is the arc length of $\Gamma(s)$. Let P be the plane tangent to $\dot{x}_o$, passing through $\Gamma(s)$ and x_o. Let $\beta(\sigma)$ be the curve on the intersection of P and $\partial\Omega$. Then, locally at t and x, we have a two dimensional visibility problem on the plane P, in which $\beta(\sigma)$ defines the boundary of the objects. Following this reasoning, κ should naturally be taken from $\beta(\sigma)$. See Figure 10.

Alternatively and more naturally under our level set formulation, we rederive the above motion law as

$$\dot{x} = \frac{II(x-x_0)}{|II(x-x_0)|^2}\left(\dot{x}_0 \cdot \frac{\nabla\phi}{|\nabla\phi|}\right), \tag{8}$$

where II is the *second fundamental form*, which can conveniently be extended to the other level sets and takes the form:

$$II = \frac{1}{|\nabla\phi|} P_{\nabla\phi} \nabla^2\phi P_{\nabla\phi}.$$

Here, $P_{\nabla\phi}$ is the orthogonal projection matrix projecting vectors to the plane with normal vector parallel to $\nabla\phi$.

For a detailed derivation and implementation, please see sections 5.4, 5.3 and 5.7. Figure 11 shows a result of horizon motion on a nonconvex body.[4]

[4]For more exmaples on the horizon motion, please see `http://www.math.ucla.edu/ ytsai/math_page`

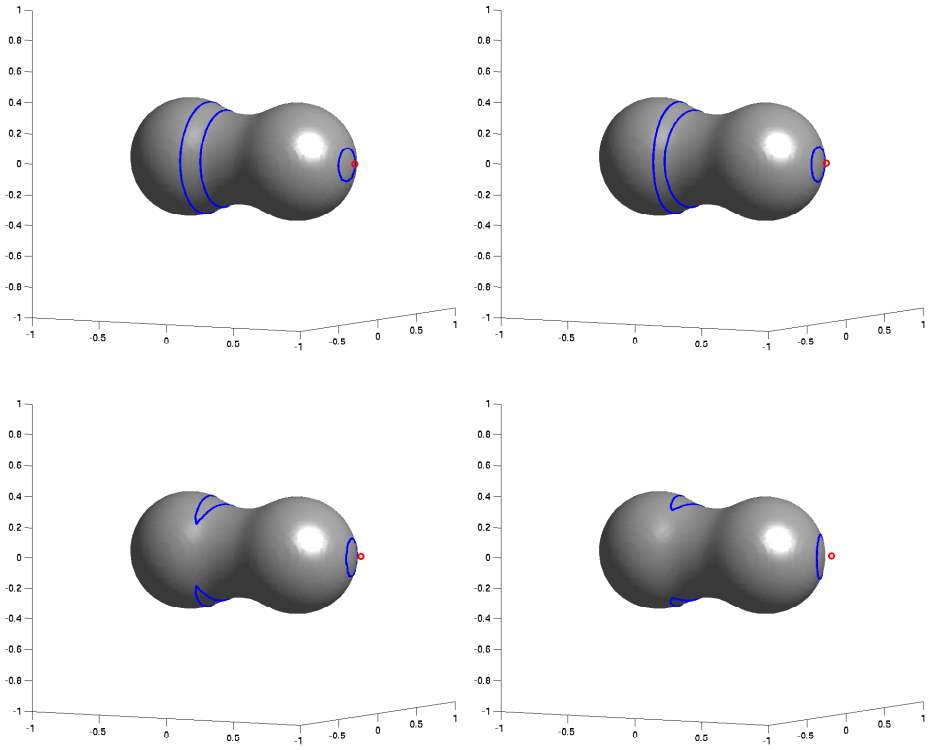


Figure 11: A example of moving horizon around a nonconvex occluder. Observe that the horizon curves break and change topology.

3.3 The dynamics of the cast horizon

Assume that $\mathbf{x}$ is a cast horizon point and $\mathbf{x}^*(\mathbf{x})$ is its generator. In two dimensions, the motion of $\mathbf{x}$ is determined by the following constraints:

$$\begin{cases} \phi(\mathbf{x}) = 0, \\ \frac{\mathbf{x}-\mathbf{x}_o}{|\mathbf{x}-\mathbf{x}_o|} = \frac{\mathbf{x}^*-\mathbf{x}_o}{|\mathbf{x}^*-\mathbf{x}_o|}. \end{cases} \tag{9}$$

Inverting, we find that the motion of the cast horizon can be written as follows:

$$\dot{\mathbf{x}} = \frac{1}{\nu \cdot \mathbf{n}(\mathbf{x})}\left(\frac{|\mathbf{r}|}{|\mathbf{r}^*|}\dot{\mathbf{r}}^* \cdot (\nu^*)^\perp + \dot{\mathbf{x}}_\mathbf{o} \cdot \nu^\perp\right)\mathbf{n}^\perp(\mathbf{x}), \tag{10}$$

where

$$\mathbf{n}^\perp(\mathbf{x}) := \begin{pmatrix} \phi_{x_2}(\mathbf{x}) \\ -\phi_{x_1}(\mathbf{x}) \end{pmatrix} / |\nabla\phi(\mathbf{x})|,$$

and similarly for $\nu^\perp$. See Section 5.5 for a detailed derivation. See Figure 12 for a computational result using this formula. We notice that these constraints also tell us how the shadow boundaries should move.

In three dimensions, we can reduce the instantaneous motion to a two dimensional problem on the "right" section of the surface following the reasoning given in the previous subsection.

Motions of the shadow boundaries

How does the shadow move in space? We can constrain a point on the shadow boundary to move only normal to the viewing direction (ergo, the shadow boundary):

$$\begin{cases} \mathbf{x}' \cdot \nu = 0, \\ \frac{\mathbf{x}-\mathbf{x}_o}{|\mathbf{x}-\mathbf{x}_o|} = \frac{\mathbf{x}^*-\mathbf{x}_o}{|\mathbf{x}^*-\mathbf{x}_o|}. \end{cases} \tag{11}$$

Motions of horizons and cast horizons of dynamic surfaces

We remark that we we are able to derive the motion laws of the visible contours even when the occluders are changing shapes. In this case, the embedding level set function ϕ is a function of space and time, $\phi(\mathbf{x}, t)$ and differentiating formulas (6) and (9) with respect to t will bring $\phi(\mathbf{x}, t)$ into the equations.

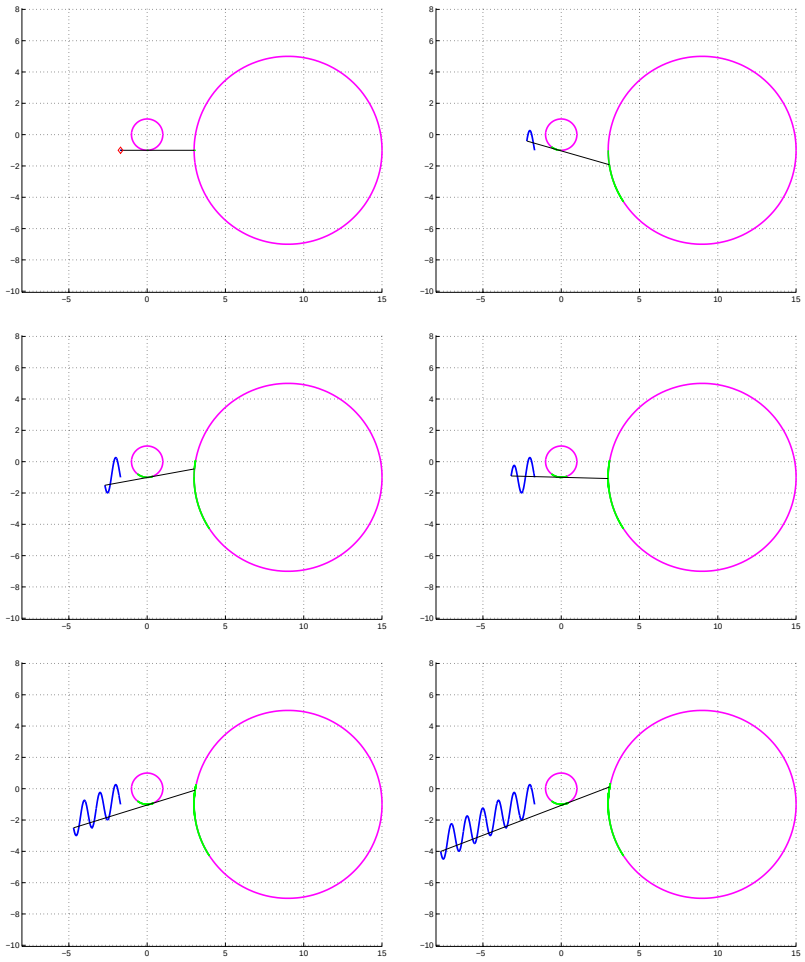


Figure 12: A result of tracking the horizon and cast horizon motion using the formulas derived in this paper. The blue curve represents the trajectory of the vantage point and the green curves represent the paths of the horizon and cast horizon. The black line links the current position of the vantage point and the cast horizon; it shows that the colinearity of the vantage point, the horizon and its cast is preserved.

3.4 Analysis of the motions

PDEs for moving the level set functions

Once we have the motion laws, we can extend the velocity to the domain near the surfaces and obtain the corresponding velocity field $\mathbf{v}(\mathbf{x})$. We then evolve the level set function(s) u in question by

$$u_t + \mathbf{v} \cdot \nabla u = 0.$$

Furthermore, the velocity fields for horizon and cast horion motions do not depend on the function u. In horizon motion, the velocity is a function of postion, time, $\dot{\mathbf{x}}_\mathbf{o}$, and the derivatives of ϕ, i.e. $\mathbf{v} = \mathbf{v}(t, \mathbf{x}, \mathbf{x_o}, \nabla\phi, D^2\phi)$. Furthermore, the level set function to be evolved is $\hbar$. In cast horizon motion, we have $\tilde{\mathbf{v}} = \tilde{\mathbf{v}}(t, \mathbf{x}, \mathbf{x_o}, \nabla\phi, D^2\phi, \hbar)$, and the level set function to be evolved is $\tilde{\psi}$. Therefore, we are evolving the following two level set equations:

$$\hbar_t + \mathbf{v}(\mathbf{t}, \mathbf{x}, \mathbf{x_o}, \nabla\phi, D^2\phi) \cdot \nabla\hbar = 0,$$

$$\tilde{\psi}_t + \tilde{\mathbf{v}}(\mathbf{t}, \mathbf{x}, \mathbf{x_o}, \nabla\phi, D^2\phi, \hbar) \cdot \nabla\tilde{\psi} = 0.$$

These are simple convection equations whose viscoity solutions are well studied, provided that the velocity fields are bounded. We only have to be careful near singularities.

Singularities in the velocity fields

Formula (7) reveals a few interesting facts. First, we notice that the speed of the horizon motion is inversely proportional to the normal curvature in the viewing direction and to the distance between the horizon and the vantage point. If the vantage point is moving in the tangent direction ν, the horizon will not move (since $\dot{\mathbf{x}}_o \cdot \mathbf{n} = 0$). The speed of the horizon motion becomes singular if the curvature of the surface at the horizon location becomes zero. On strictly convex objects, this will never happen. If we restrict our analysis to a single connected smooth non-convex object, we will see easily that at the instance in which a horizon point moves into the location where $\kappa = 0$, a neightborhood of this location becomes completely visible. This signifies the disappearance of the horizon point. If the course of the vantage point is reversed, we get the genesis of a new horizon point.

Formula (10) tells us that the motion of a cast horizon point becomes singular when it is a horizon point ($\nu \cdot \mathbf{n} = 0$). On a single non-convex smooth surface, this happens precisely when a horizon point and its cast across the concavity collide into each other at the location where $\kappa = 0$. In the setting where there are multiple

strictly convex objects, this also describes the changing of the cast horizon into a visible horizon point which is previously invisible. Therefore, the singularities of the horizons and cast horizons describe a part of their genesis. A complete genesis of the visible contours includes another part, in which a hidden object suddenly becomes visible. We shall discuss this point in a later subsection.

3.5 Relating horizon and its shadow

To move the cast horizon, following the notation used in the previous section, we need to find $\mathbf{x}^*(\mathbf{x})$ for each point $\mathbf{x}$ on the cast horizon.

3.5.1 Explicit formula

$\mathbf{x}$ and $\mathbf{x}^*(\mathbf{x})$ are related by

$$\mathbf{x}^*(\mathbf{x}) := \mathbf{x} - r(\mathbf{x})\nu(\mathbf{x});$$

$r(\mathbf{x})$ can be computed by

$$r(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_o| - \rho(\nu(\mathbf{x})),$$

where ν and ρ are defined as previously.

3.5.2 Implicit formulation

We follow the spirit of formula (2) introduced in Section 2 and propagate the link between the horizon and its cast shadow implicitly. Define $\vartheta : \mathbb{R}^d \mapsto \mathbb{R}^d$ to be

$$\vartheta(\mathbf{x}) := \begin{cases} \mathbf{x} & \text{if } \tilde{\psi}(\mathbf{x}) = \phi(\mathbf{x}) \\ \mathbf{x}' & \text{if } \tilde{\psi}(\mathbf{x}) = \tilde{\psi}(\mathbf{x}'). \end{cases}$$

$\tilde{\psi}$ is the function defined in Section 3.1. When we move the points $\mathbf{x}$ near the cast horizon, we also move the points $\vartheta(\mathbf{x})$, which are points near the horizon. By continuity around the cast horizon, we will have the right motion of the cast horizon.

3.6 Reinitialization and emergence-time estimate

It can be easily seen from figure 13 that a completely hidden object may suddenly become visible at a later time during the journey of the vantage point. At the time of emergence, we need to reinitialize our algorithm, i.e., we need to find the apparent contours on the newly emerged surfaces to get the correct visibility information.

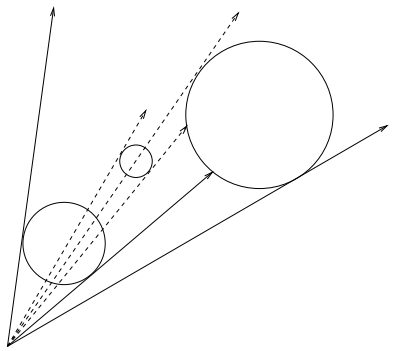


Figure 13: Model scenario I

An explicit reinitialization criterion

Assuming that we are merely tracking the visibility boundaries on the objects. How do we know when to initialize? We can formalize the reasoning as follows. We define the map $\mathcal{G}^{-1} : S^{d-1} \mapsto \{\mathbf{x}_i\}_{i=0}^{N(\theta)}$ such that

$$\mathcal{G}^{-1}(\theta) := \{\mathbf{x}_i \in \mathbb{R}^d : \nabla\phi(\mathbf{x}_i)/|\nabla\phi(\mathbf{x}_i)| = \theta \text{ and } \phi(\mathbf{x}_i) = 0\}.$$

This map is the inverse of the Gauss map in the case that $\{\phi = 0\}$ is a strictly convex hypersurface. Let $\mathcal{S}$ be the set containing x and x^*. We reinitialize whenever there exists an $x \in S$ such that $\exists\, y \in \mathcal{G}(\nu(\mathbf{y}))$ with $\mathbf{x}^*(\mathbf{x}) \prec y \prec \mathbf{x}$.

This provides an explicit criterion for reinitialization. However, we can do better with our implicit formulation. This amounts to knowing 1) how the shadow moves 2) how far a hidden surface is from the shadow boundaries.

Emergence-time estimate

Given current vantage point position and its motion, we want to estimate the emergence time for an object that is occluded. We begin by assuming the the curvatures of the surfaces locally around the regions of interest are constant. The diagram in Figure 14 shows a model configuration: The small circle is initially occluded by the larger circle on the left. We want to estimate the time interval δt between this instance t_0 and the time $t_1 = t_0 + \delta t$ when the small circle first emerges into the scene.

Following the discussion above, consider a point $\mathbf{y}$ on the shadow boundaries away from the horizon such that $\nabla\phi(\mathbf{y}) \perp \nu(\mathbf{y})$ and $\nabla\phi(\mathbf{y}) \cdot \nabla\phi(\mathbf{y}(\mathbf{x})) > 0$. This is the point closest to some hidden part of the objects.

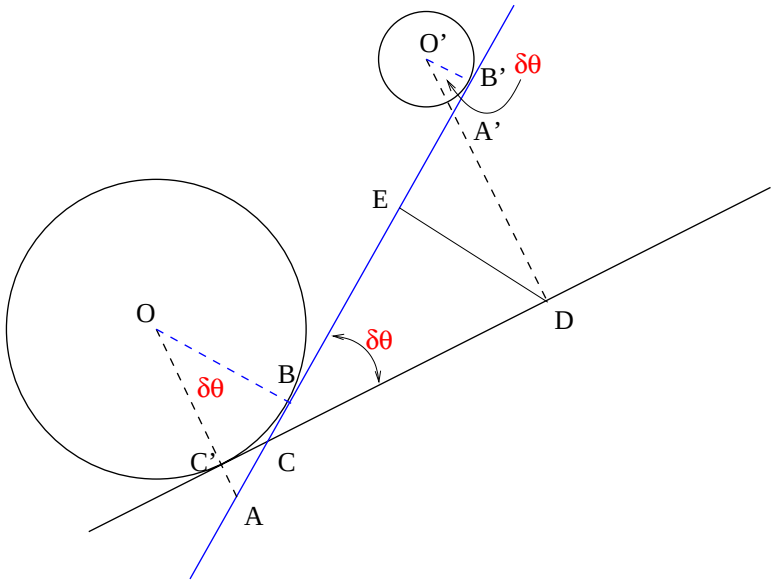


Figure 14: A diagram for emergence time estimate.

For consistency of notation, we will use D in place of $\mathbf{y}$. Let d be the distance between D and the circle centered at O'. Let ρ and ρ' be the radius of the circle centered at O and O' respectively. Let r denote the distance between D and C'. By elementary Euclidean geometry, we have the following identities:

$$CC' = \rho \tan\frac{\delta\theta}{2}, \;\; CD = r - CC' = r - \rho\tan\frac{\delta\theta}{2}, \;\; \Longrightarrow DE = CD\sin\delta\theta,$$

$$O'A' = \frac{1}{\cos\delta\theta}\rho'.$$

Therefore, we can find $\delta\theta$ from the last two equalities. Since we know how fast the horizon is moving, we can then determine δt.

Let $\dot{\mathbf{x}} = |\dot{\mathbf{x}}|_\infty$,

$$\frac{DE}{\delta t} = |\dot{\mathbf{x}}|_\infty, \;\; \delta t = \frac{DE}{|\dot{\mathbf{x}}|_\infty}.$$

Further considerations

We have mentioned in the beginning of this paper that the approach of moving the visible contour may not be more efficient than simply performing implicit ray tracing in general "large world" configurations. Here we construct such a case to

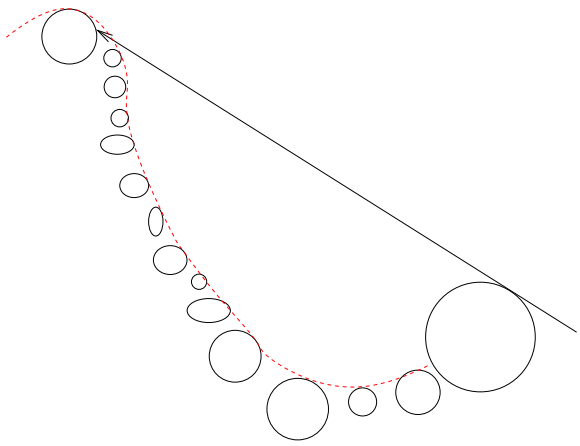


Figure 15: bad case for moving curves

validate our arguments. Consider a nonconvex part of an object as shown by the dashed red curve in Figure 15. Suppose the red curve is broken down into dense small disconnected components. In the case where the red curve is the nonconvex part of a connected component and with the viewing direction being depicted in Figure 15, the cast horizon will move continuously on the nonconvex part of the object without need for reinitialization. However, in the second case, we have to reinitialize very often because the cast horizon will "jump" from component to component.

4 Conclusion and future directions

In this article, we introduced a fast implicit ray tracing algorithm independent of grid geometry and easily parallelizable. This is then extended to a multi-resolution algorithm for near optimal efficiency. Furthermore, we showed that the implicit framework captures accurately the shadow boundaries, which include the horizon and cast horizon curves. We studied how these objects move when the source point is moving. Explicit formulas which reveal the relations between the motions and the local/global geometry of the given configuration are derived and are tightly coupled with our level set framework for implementation. Also, questions such as "how soon will this hidden object appear" can be answered as a result of our algorithm.

There is a rich pool of applications related to the visibility problem described in this paper. Currently we are working on problems related to navigation, visibility with occluders changing shapes in time, in non-uniform media. Our solutions

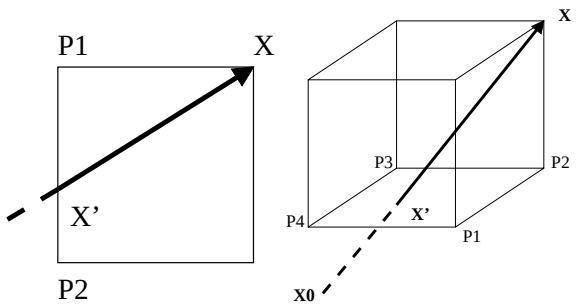


Figure 16: A demonstration of 2D and 3D interpolation

will combine approaches both from the PDE formulation and the algorithms in computational geometry.

Acknowledgement

The author YT wishes to thank the organizers of IPAM Geometrically Based Motion Workshop and IPAM staff.

5 Appendix

5.1 Interpolation schemes

Since the majority of visibility applications benefit from the simplicity of Cartesian grids, we need to adapt the algorithm in order to take advantage of this. As described in the algorithm, at each grid point $x = (i, j, k)$, we need to determine an upwind neighbor x' and find the value of $\psi(x')$. In most cases, x' does not lie on the grid. Therefore, we need to interpolate the values of ψ from the grid points closest to x'. For simplicity and speed considerations, we choose to perform linear interpolation in 2D and bilinear interpolation in 3D. In Figure 16, we use $\psi(P_1)$ and $\psi(P_2)$ for linear interpolation in the 2D case and use $\psi(P_i)$, $i = 1, 2, 3, 4$, for bilinear interpolation of ψ.

We note that a fast marching or fast sweeping strategy for determining distance from the source point and passing values can be used in place of this interpolation.

Let ψ_{int} be the interpolant near x', we know that $\psi_{\text{int}}(x') = \psi(x') + \mathcal{O}(h^2)$. Thus, the discrete visibility equation (2) is in effect

$$\psi(x) = \min(\psi_{\text{int}}(x'), \phi(x)).$$

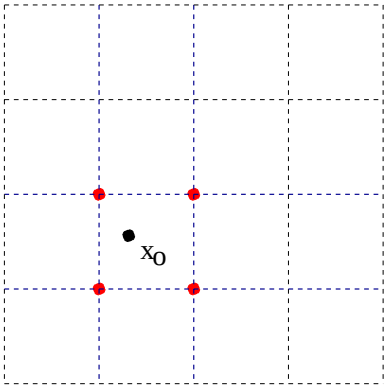


Figure 17: The red points denote the cell vertices.

5.2 Examples of star-shaped updating sequence (sweeping)

There are many different ways of implementing a star-shaped updating sequence. One approach is to use the algorithm based on the heap sort strategy [25] to find grid nodes for update based on their distance to the vantage point. However, due the complexity involved with heap sort, this algorithm is not optimal.

Alternatively, we use a sweeping approach in our simulation. For example, let us consider a Cartesian grid in 2D and assume that the vantage point lies on a grid node; we can then consider separately the visibility problem in each of the four quadrants centered at the vantage point. For simplicity, let us assume that the vantage point is at the origin and the grid is represented by the lattice $[-n_x, n_x] \times [-n_y, n_y] \subset \mathbb{Z}^2$. A compact way of writing this sweeping sequence in C/C++ is:

```
for(s1=-1;s1<=1;s1+=2)
for(s2=-1;s2<=1;s2+=2)
for(i=0;(s1<0?i>=-nx:i<=nx);i+=s1)
for(j=0;(s2<0?j>=-ny:j<=ny);j+=s2)
  update ψ_{i,j}.
```

In the case where $\mathbf{x_o}$ does not lie on a grid node, we describe an easy modification to the updating sequence above. Let $\mathbf{x_o} \in I_o := [x_{i_0}, x_{i_0+1}) \times [y_{j_0}, y_{j_0+1})$. Update the values of ψ on the vertices of I_o. Then update the grid nodes in the strips $\{(x_i, y_j) : i = i_0, i_0 + 1 \text{ and } j = -n_y \text{ to } n_y\}$ and $\{(x_i, y_j) : i = -n_x, n_x \text{ and } j = j_0, j_{0+1}\}$. Finally, update the remaining four quadrants independently. See Figure 17 for a depiction of this approach.

5.3 Finding the curvature of a specified direction

As we argued in Section 3.2, the three dimensional problem of determining the motion of the horizon can be reduced to an instantaneous two dimensional problem. In order to move the horizon in this manner, we need to evaluate the curvature of the surface in the specified direction. Here we present a way to do that.

Let τ be the tangent vector being specified. We want to find the curvature on $\partial\Omega$ in this direction. First let $p(\mathbf{x};\tau)$ be the plane passing through $\mathbf{x}$, spanned by $\mathbf{n}(x)$ and τ, and let P be the level set function that embeds this plane. Then

$$\tilde{\tau} = \frac{\nabla\phi \times \nabla P}{|\nabla\phi \times \nabla P|},$$

where $\tilde{\tau}(\mathbf{x}) = \tau$, and the curvature is

$$k_\tau \mathbf{n} = \vec{\nabla}\tilde{\tau} \cdot \tilde{\tau}.$$

5.4 Derivation of the dynamics of horizon

We follow the constraints (6):

$$\begin{cases} \phi(\mathbf{x}) = 0 \\ (\mathbf{x} - \mathbf{x}_0) \cdot \nabla\phi(\mathbf{x}) = 0 \end{cases}$$

and differentiate with respect to t, we have

$$\nabla\phi(\mathbf{x}) \cdot \dot{\mathbf{x}} = 0 \tag{12}$$

$$(\mathbf{x} - \mathbf{x}_o) \cdot D^2\phi(\mathbf{x})\dot{\mathbf{x}} = \dot{\mathbf{x}}_o \cdot \nabla\phi(\mathbf{x}) \tag{13}$$

In 2 space dimensions, these two relations uniquely determine the motion of x with given initial conditions. Writing $(\mathbf{x} - \mathbf{x}_o) = |\mathbf{x} - \mathbf{x}_o|\, n^{\perp}(\mathbf{x}) = |\mathbf{x} - \mathbf{x}_o|\,(-\phi_y, \phi_x)/|\nabla\phi|$, we have

$$\begin{aligned} &\frac{|\mathbf{x} - \mathbf{x}_o|}{|\nabla\phi|} \begin{pmatrix} -\phi_y \\ \phi_x \end{pmatrix} \cdot \begin{pmatrix} \phi_{xx} & \phi_{xy} \\ \phi_{yx} & \phi_{yy} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \\ &= \frac{|\mathbf{x} - \mathbf{x}_0|}{|\nabla\phi|} \begin{pmatrix} -\phi_y\phi_{xx} + \phi_x\phi_{yx} \\ -\phi_y\phi_{xy} + \phi_x\phi_{yy} \end{pmatrix} \cdot \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}, \end{aligned}$$

and

$$\begin{aligned} \frac{|\mathbf{x} - \mathbf{x}_o|}{|\nabla\phi|} \begin{pmatrix} \phi_x & \phi_y \\ -\phi_y\phi_{xx} + \phi_x\phi_{yx} & -\phi_y\phi_{xy} + \phi_x\phi_{yy} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \\ = \begin{pmatrix} 0 \\ \dot{\mathbf{x}}_o \cdot \nabla\phi(\mathbf{x}) \end{pmatrix} \end{aligned}$$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \frac{|\nabla\phi|}{|\mathbf{x}-\mathbf{x}_o|}\frac{1}{D}\begin{pmatrix} -\phi_y\phi_{xy}+\phi_x\phi_{yy} & -\phi_y \\ \phi_y\phi_{xx}-\phi_x\phi_{yx} & \phi_x \end{pmatrix}\begin{pmatrix} 0 \\ \dot{\mathbf{x}}_o\cdot\nabla\phi(\mathbf{x}) \end{pmatrix}$$
$$= \frac{|\nabla\phi|}{|\mathbf{x}-\mathbf{x}_o|}\frac{\dot{\mathbf{x}}_o\cdot\nabla\phi(\mathbf{x})}{D}\begin{pmatrix} -\phi_y \\ \phi_x \end{pmatrix},$$

where

$$\begin{aligned} D &= \det\begin{pmatrix} \phi_x & \phi_y \\ -\phi_y\phi_{xx}+\phi_x\phi_{yx} & -\phi_y\phi_{xy}+\phi_x\phi_{yy} \end{pmatrix} \\ &= -\phi_x\phi_y\phi_{xy}+\phi_x^2\phi_{yy}+\phi_y^2\phi_{xx}-\phi_x\phi_y\phi_{xy} \\ &= \phi_x^2\phi_{yy}+\phi_y^2\phi_{xx}-2\phi_x\phi_y\phi_{xy}. \end{aligned}$$

Since the curvature of $\partial\Omega$ at x is

$$\begin{aligned} \kappa &= \nabla\cdot\frac{\nabla\phi}{|\nabla\phi|} \\ &= \frac{1}{|\nabla\phi|^3}(\phi_x^2\phi_{yy}+\phi_y^2\phi_{xx}-2\phi_x\phi_y\phi_{xy}), \end{aligned}$$

the motion of x is

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \frac{1}{\kappa}\frac{\dot{x}_o\cdot n(\mathbf{x})}{|\mathbf{x}-\mathbf{x}_o|}n^{\perp}(\mathbf{x}). \tag{14}$$

We define $n^{\perp}(\mathbf{x}) = (\mathbf{x}-\mathbf{x}_o)/|\mathbf{x}-\mathbf{x}_o|$.

Alternatively, we can write $\dot{x}$ in a slightly different form:

$$\dot{x} = \frac{P_{\nabla\phi}\nabla^2\phi(x-x_0)}{|P_{\nabla\phi}\nabla^2\phi(x-x_0)|^2}(\dot{x}_0\cdot\nabla\phi),$$

where P_v is the orthogonal projection matrix projecting vectors to the plane with normal vector v. Let us check this expression for the velocity of the curve. Note $\nabla\phi\cdot\dot{x}=0$ since $v\cdot P_v w = 0$ for all vectors v and w. Also, $\nabla^2\phi(x-x_0)\cdot\dot{x} = \nabla\phi\cdot\dot{x}_0$ is satisfied since $P_v w\cdot w = |P_v w|^2$ for all vectors v and w. Thus this velocity is valid and is the first form in our alternate derivation.

Geometric interpretation

If x indeed represents the position of the curve, then $x-x_0$ is tangent to the object surface at x and so $x-x_0 = P_{\nabla\phi}(x-x_0)$. Making this replacement above gives our second form for the velocity,

$$\dot{x} = \frac{P_{\nabla\phi}\nabla^2\phi P_{\nabla\phi}(x-x_0)}{|P_{\nabla\phi}\nabla^2\phi P_{\nabla\phi}(x-x_0)|^2}(\dot{x}_0\cdot\nabla\phi).$$

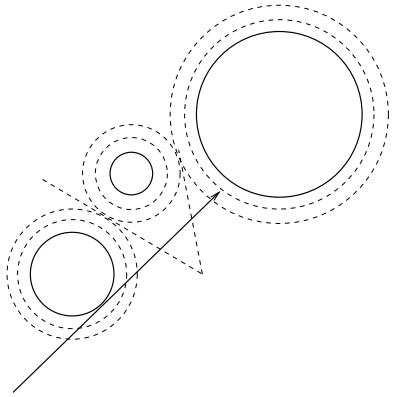


Figure 18: Model cast horizon scenario

This form is particularly nice because we know the second fundamental form in a level set framework is transformed to

$$II = \frac{1}{|\nabla\phi|} P_{\nabla\phi} \nabla^2 \phi P_{\nabla\phi}.$$

Thus we can rewrite the velocity in its final form,

$$\dot{x} = \frac{II(x - x_0)}{|II(x - x_0)|^2} \left(\dot{x}_0 \cdot \frac{\nabla\phi}{|\nabla\phi|} \right).$$

IIv evaluated at a point represents the change in the normals of the object surface in the direction of v at that point.

5.5 Derivation of the dynamics of the cast horizon

We assume that the level sets of ϕ near $\mathbf{x}$ are smooth curves and are not tangent to ν. In two space dimension, we have two equations that determine the dynamics of $\mathbf{x}$:

$$\begin{cases} \phi(\mathbf{x}) = \text{constant} \\ \nu = \tilde{\nu} \end{cases} \tag{15}$$

Let $\mathbf{r}$ and $\tilde{\mathbf{r}}$ denote $(\mathbf{x} - \mathbf{x_o})$ and $(\tilde{\mathbf{x}} - \mathbf{x_o})$ respectively. Differentiating these equations, we arrive at:

$$\nabla\phi(\mathbf{x}) \cdot \mathbf{x}' = 0,$$

$$\frac{\mathbf{r}'}{|\mathbf{r}|} - \frac{\mathbf{r}}{|\mathbf{r}|^2} \nu \cdot \mathbf{r}' = \frac{\tilde{\mathbf{r}}'}{|\tilde{\mathbf{r}}|} - \frac{\tilde{\mathbf{r}}}{|\tilde{\mathbf{r}}|^2} \tilde{\nu} \cdot \tilde{\mathbf{r}}'.$$

Notice that the term

$$\nu \cdot \mathbf{r}' \frac{\mathbf{r}}{|\mathbf{r}|} = (\nu \cdot \mathbf{r}')\nu$$

is $\mathbf{r}'$ projected onto the unit vector ν. Therefore, the left hand side denotes the projection of $\mathbf{r}'$ onto the unit vector $\nu^\perp$:

$$\frac{1}{|\mathbf{r}|}(\mathbf{r}' - \nu \cdot \mathbf{r}'\nu) = \frac{\mathbf{r}' \cdot \nu^\perp}{|\mathbf{r}|} = \frac{1}{|\mathbf{r}|} P_\nu \mathbf{r}'.$$

Similarly, with the right hand side, we have the equation:

$$\frac{1}{|\mathbf{r}|} P_\nu \mathbf{r}' = \frac{1}{|\tilde{\mathbf{r}}|} P_{\tilde{\nu}} \tilde{\mathbf{r}}'.$$

Keeping in mind that we want to solve for $\mathbf{x}'$, we move every other term to the right hand side and arrive at

$$P_\nu \mathbf{x}' = \frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} P_{\tilde{\nu}} \tilde{\mathbf{r}}' + P_\nu \mathbf{x}'_{\mathbf{o}},$$

$$\nabla \phi(\mathbf{x}) \cdot \mathbf{x}' = 0.$$

In two dimensions, $P_\nu w = (w \cdot \nu^\perp)\nu^\perp$**,** and $P_\nu w \cdot \nu^\perp = w \cdot \nu^\perp$, therefore, we have

$$\begin{pmatrix} \nu_2 & -\nu_1 \\ \phi_{x_1} & \phi_{x_2} \end{pmatrix} \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} \frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' \cdot v^\perp + \mathbf{x}'_{\mathbf{o}} \cdot \nu^\perp \\ 0 \end{pmatrix},$$

and consequently,

$$\begin{aligned} \begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} &= \frac{1}{\nu \cdot \nabla \phi(\mathbf{x})} \cdot \\ &\quad \begin{pmatrix} \phi_{x_2} & \nu_1 \\ -\phi_{x_1} & \nu_2 \end{pmatrix} \begin{pmatrix} \frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' \cdot \nu^\perp + \mathbf{x}'_{\mathbf{o}} \cdot \nu^\perp \\ 0 \end{pmatrix} \\ &= \frac{1}{\nu \cdot \nabla \phi(\mathbf{x})} \\ &\quad (\frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' \cdot \nu^\perp + \mathbf{x}'_{\mathbf{o}} \cdot \nu^\perp) \begin{pmatrix} \phi_{x_2} \\ -\phi_{x_1} \end{pmatrix} \\ &= \frac{\mathbf{n}^\perp(\mathbf{x})}{\nu \cdot \mathbf{n}(\mathbf{x})} (\frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' + \mathbf{x}'_{\mathbf{o}}) \cdot \nu^\perp \\ &= \frac{(\frac{|\mathbf{r}|}{|\tilde{\mathbf{r}}|} \tilde{\mathbf{r}}' + \mathbf{x}'_{\mathbf{o}}) \cdot \nu^\perp}{\nu \cdot \mathbf{n}(\mathbf{x})} \mathbf{n}^\perp(\mathbf{x}) \end{aligned}$$

where

$$\nabla^\perp \phi(\mathbf{x}) := \begin{pmatrix} \phi_{x_2} \\ -\phi_{x_1} \end{pmatrix}, \text{ and } n^\perp := \frac{\nabla^\perp \phi(\mathbf{x})}{|\nabla^\perp \phi(\mathbf{x})|}.$$

5.6 An algorithm to project $\{\psi = 0\}$ to S^{n-1}

We discretize S^{n-1} over a Cartesian grid: $\rho : [0, 2\pi) \times [-\pi, \pi) \mapsto x_o + S^{n-1}/r$, with r large enough so that $x_o + S^{n-1}/r$ lies completely outside of ψ. $\Delta\theta_1$, $\Delta\theta_2$ are set to be the smallest angle possible on the grid that discretizes ψ. Therefore, we can set $\Delta\theta_1 = \tan^{-1}(\min(\Delta x, \Delta y)/L)$, where L is the diagonal length of the grid.

For each $\rho_{l,m}$, we then shoot a ray, starting from $p_0 = \rho_{l.m}$ outward according to the angle determined by (l, m), and use the bisection method to find p_1, the intersection of the ray and $\{\psi = 0\}$.

Note that the complexity is $N^2 \log N$ for a grid of size N^3.

5.7 Numerics

We computed the quantities describe in this paper using standard level set technologies. Please refer to [19, 20, 21, 22, 10] for details.

5.8 A list of level set functions used in this paper

We provide a comprehensive list of the level set functions we construct in this paper:

ϕ embeds the objects

$\hbar(\mathbf{x}) := (\mathbf{x} - \mathbf{x_o}) \cdot \nabla\phi(\mathbf{x})$ characterizes the horizon

$\tilde{\phi} := \max(\phi, -\hbar)$ $\{\tilde{\phi} \leq 0\} = \{\phi \leq 0\} \setminus \{\hbar < 0\}$, defines the same visibility as ϕ

ψ the visibility map resulting from the implicit ray tracing on ϕ

$\tilde{\psi}$ the visibility map resulting from the implicit ray tracing on $\tilde{\phi}$, characterizes the cast horizon

ϑ: $\mathbb{R}^d \mapsto \mathbb{R}^d$ links horizon to its cast implicitly

References

[1] D. Adalsteinsson and J.A. Sethian. An overview of level set methods for etching, deposition, and lithography development. *IEEE Transactions on Semiconductor Devices*, 10(1), February 1997.

[2] Pankaj K. Agarwal and Micha Sharir. Ray shooting amidst convex polygons in 2D. *J. Algorithms*, 21(3):508–519, 1996.

[3] Pankaj K. Agarwal and Micha Sharir. Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions. *SIAM J. Comput.*, 25(1):100–116, 1996.

[4] Luis Alvarez, Frédéric Guichard, Pierre-Louis Lions, and Jean-Michel Morel. Axioms and fundamental equations of image processing. *Arch. Rational Mech. Anal.*, 123(3):199–257, 1993.

[5] M. Bertalmio, G. Sapiro, and G. Randall. Dynamic visibility in an implicit framework. *IEEE Trans. Medical Imaging*, 18:448–451, 1999.

[6] Marcelo Bertalmio, Li-Tien Cheng, Stanley Osher, and Guillermo Sapiro. Variational problems and partial differential equations on implicit surfaces. *J. Comput. Phys.*, 174(2):759–780, 2001.

[7] M. D. Betterton. Theory of structure formation in snowfields motivated by penitentes, suncups, and dirt cones. *Physical Review E*, 63, 2001.

[8] Paul Burchard, Li-Tien Cheng, Barry Merriman, and Stanley Osher. Motion of curves in three spatial dimensions using a level set approach. *J. Comput. Phys.*, 170:720–741, 2001.

[9] J. C. Carr, R. K. Beatson, J.B. Cherrie T. J. Mitchell, B. C. McCallum W. R. Fright, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH*, 2001.

[10] Li-Tien Cheng, Paul Burchard, Barry Merriman, and Stanley Osher. Motion of curves constrained on surfaces using a level set approach. *J. Comput. Phys.*, 175:604–644, 2002.

[11] Roberto Cipolla and Peter Giblin. *Visual motion of curves and surfaces*. Cambridge University Press, 2000.

[12] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *ACM Symposium on Interactive 3D Graphics*, 1997.

[13] Fredo Durand. 3d visibility: Analysis study and applications. *PhD Thesis, MIT*, 1999.

[14] Fredo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. In *SIGGRAPH*, 2000.

[15] Memoli Facundo and Guillermo Sapiro. Fast computation of distance functions and geodesics on implicit surfaces. To appear, Journal of Computational Physics (2001).

[16] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *SIGGRAPH*, 2001.

[17] Hailin Jin, Anthony Yezzi, and Stefano Soatto. Stereoscopic shading: Integating multi-frame shape cues in a variational framework. *In Preparation*.

[18] Stanley Osher, Li-Tien Cheng, Myungjoo Kang, Hyeseon Shim, and Yen-Hsi Tsai. Geometric optics in a phase space based level set and eulerian framework. 2001. Submitted to Journal of Computational Physics.

[19] Stanley Osher and Ronald P. Fedkiw. Level set methods: an overview and some recent results. *J. Comput. Phys.*, 169(2):463–502, 2001.

[20] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.

[21] Stanley Osher and Chi-Wang Shu. High-order essentially nonoscillatory schemes for Hamilton-Jacobi equations. *SIAM J. Numer. Anal.*, 28(4):907–922, 1991.

[22] Danping Peng, Barry Merriman, Stanley Osher, Hongkai Zhao, and Myungjoo Kang. A PDE-based fast local level set method. *J. Comput. Phys.*, 155(2):410–438, 1999.

[23] G. Sapiro, R. Kimmel, D. Shaked, B. B. Kimia, and A. M. Bruckstein. Implementing continuous scale morphology via curve evolution. *Pattern Recognition*, 26(9):1363–1372, 1993.

[24] J. A. Sethian. *Level set methods and fast marching methods*. Cambridge University Press, Cambridge, second edition, 1999. Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science.

[25] John Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[26] Hong-Kai Zhao, Stanley Osher, Barry Merriman, and Myungjoo Kang. Implicit and non-parametric shape reconstruction from unorganized points using variational level set method. *Computer Vision and Image Understanding*, 80:295–319, 2000.

3D Scan Conversion of CSG Models into Distance Volumes

David E. Breen[1] Sean Mauch[1] Ross T. Whitaker[2]

[1]California Institute of Technology
[2]University of Tennessee, Knoxville

Abstract

A distance volume is a volume dataset where the value stored at each voxel is the shortest distance to the surface of the object being represented by the volume. Distance volumes are a useful representation in a number of computer graphics applications. In this paper we present a technique for generating a distance volume with sub-voxel accuracy from one type of geometric model, a Constructive Solid Geometry (CSG) model consisting of superellipsoid primitives. The distance volume is generated in a two step process. The first step calculates the shortest distance to the CSG model at a set of points within a narrow band around the evaluated surface. Additionally, a second set of points, labeled the zero set, which lies on the CSG model's surface are computed. A point in the zero set is associated with each point in the narrow band. Once the narrow band and zero set are calculated, a *Fast Marching Method* is employed to propagate the shortest distance and closest point information out to the remaining voxels in the volume. Our technique has been used to scan convert a number of CSG models, producing distance volumes which have been utilized in a variety of computer graphics applications, e.g. CSG surface evaluation, offset surface generation, and 3-D model morphing.

1 Introduction

Volume graphics is a growing field which generally involves representing three dimensional objects as a rectilinear 3-D grid of scalar values, a volume dataset. Given this kind of representation numerous algorithms have been developed to process, manipulate and render volumes. Volume datasets may be generated in a variety of ways. Certain scanning devices, e.g. MRI and CT, generate a rectilinear grid of scalar values directly from their scanning process. The scalar values can represent the concentration of water or the density of matter at each grid point (voxel). Additionally, volume datasets can be generated from conventional geometric models, using a process called 3-D scan conversion.

When 3-D scan converting a geometric model to a volumetric representation it is not always clear what value should be stored at each voxel of the volume, and what that value should represent. Here, we propose the use of distance volumes. A distance volume is a volume dataset where the value stored at each voxel is the shortest distance to the surface of the object being represented by the volume. If the object is closed, a signed distance may be stored to provide additional inside-outside information. We store negative values inside the object and positive distances outside. In this paper we will show how to generate a distance volume with sub-voxel accuracy from one type of geometric model, a Constructive Solid Geometry (CSG) model, and we will also show that this type of volume representation is useful in a number of computer graphics applications, namely CSG surface evaluation, offset surface generation, and 3-D model morphing.

Constructive Solid Geometry (CSG) modeling is a well-developed technique that combines simple solid primitives using spatial boolean operations to produce complex three dimensional objects [15]. Some of the most commonly used primitives in CSG modeling are quadrics, superquadrics [1], and closed polygonal objects. These primitives can be added, subtracted, or intersected with each other to create a variety of solid geometric models. The structure that is used to represent a CSG model is ordinarily a binary tree. The leaf nodes of the tree contain solid primitives, superellipsoids in our case. A boolean operation is associated with each non-leaf node and a transformation matrix is associated with each arc of the tree. The CSG binary tree may also be derived from a directed acyclic graph.

While Constructive Solid Geometry is a powerful modeling paradigm, unfortunately its modeling representation cannot be directly displayed on today's graphics workstations. Additionally, it is a representation not suitable for many other types of modeling operations. Frequently the CSG tree or graph must first be evaluated and converted into a polygonal surface before it can be interactively displayed, processed or manipulated. We have found that first scan converting the CSG model into a distance volume allows us to perform several types of graphics operations on the model. Applying the Marching Cubes algorithm [12] to the distance volume and extracting the iso-surface at value zero produces a polygonal surface which approximates the evaluated CSG model. Extracting an iso-surface at a value other than zero produces offset surfaces to the CSG model. The distance volume may also be used to perform 3-D

[1]Computer Graphics Laboratory, Caltech MS 348-74, Pasadena, CA 91125, {david,sean}@gg.caltech.edu

[2]330 Ferris Hall, Department of Electrical Engineering, University of Tennessee, Knoxville, TN 37996-2100, rtw@utk.edu

model morphing. A deformable implicit model can utilize the distance information to change from one shape into another [21]. The derivative of the distance volume describes a field which effectively points in the direction of the embedded object's surface. Given an initial object and the distance volume representing a second object, forces are applied on each point of the initial model which push it towards the second object.

A distance volume is generated in a two step process. The first step calculates the shortest distance to the CSG model at a set of points within a narrow band around the evaluated surface. Additionally, a second set of points lying on the CSG model's surface, labeled the zero set, are computed. A point in the zero set is associated with each point in the narrow band. The narrow band and zero set are calculated with a modified version of the Constructive Cubes algorithm [3]. Once the narrow band and zero set are calculated, a *Fast Marching Method* similar to Sethian's [17], is employed to propagate the shortest distance and closest point information out to the remaining voxels in the volume. Sethian's approach has been used in the past to numerically solve partial differential equations, but we have modified it to use a heuristic rule for propagating closest point information instead of calculating distance with a finite difference scheme. The accuracy of our method depends on a discretization of the surface (resolution of the zero set) and is independent of the volume grid spacing. We therefore are able to calculate shortest distance at resolutions greater than the resolution of the final distance volume.

The original Constructive Cubes algorithm was developed to produce a polygonal approximation to a CSG model's surface. This is accomplished by first converting the CSG model into a volumetric representation, where the value stored at each voxel is a combination of the value of the inside-outside function for each of the model's primitives (superellipsoids) evaluated at the (x, y, z) location of the voxel. The inside-outside function of a superellipsoid is a non-linear function of (x, y, z), and is defined to be one on the surface of the primitive, less than one and greater than zero inside, and greater than one outside. The modifications made to the Constructive Cubes algorithm were designed to produce the initial closest point information near the CSG model's surface needed for the Fast Marching Method, which then calculates the shortest distance at the voxels away from the CSG model.

The first modification involves calculating the closest point to a single superellipsoid primitive. In general this is accomplished with an iterative minimization scheme. Given the closest points to separate geometric primitives (and therefore the shortest distances), a new set of combinations rules are applied to merge the distance values of the individual primitives to produce the closest point and shortest distance to the entire CSG model. Unfortunately, there are small regions near the CSG model where the combination rules generate invalid results, calculating a closest point which does not lie on the evaluated surface. These cases, which occur less than 1 percent of the time, can be easily detected and discarded by evaluating the closest points with the original Constructive Cubes algorithm.

The remainder of the paper first presents related work in 3-D scan conversion. The paper then details the steps required to produce a distance volume: generating the narrow band of points near the CSG model surface and zero set on the surface, followed by the propagation of the closest point information into the remaining voxels of the volume with the Fast Marching Method. The final section presents the results of our 3-D scan conversion method within three applications: CSG surface evaluation, offset surface generation, and 3-D model morphing.

2 Previous Work

3-D scan conversion takes a 3-D geometric model, a surface in 3-D or a solid model, and converts it into a 3-D volume data set [4, 9, 10, 11, 18], where voxels that contain the original surface or solid have a value of one. The remaining voxels have a value of zero. Using the volume-sampling methods of Wang and Kaufman [20] aliasing artifacts may be significantly reduced. These methods produce voxels with values between zero and one, where non-integer values represent voxels partially occupied by the original object. Scan converted primitives may then be rendered, or combined using CSG operations [7] with other scan converted primitives or acquired volume datasets. Payne and Toga [13] present a method for calculating distance volumes from a polygonal model. They use the distance volumes to perform a variety of surface manipulation tasks. Extensions to discrete distance transforms [2, 5], e.g. Chamfer methods, were considered for our work. They were deemed insufficient for our needs, because they do not provide sub-voxel accuracy.

Our algorithm differs from previous efforts to 3-D scan convert CSG models because we evaluate the parametric primitives directly and combine the results in object space, before scan conversion. This avoids the sampling errors produced when performing CSG operations on scan converted primitives, that are seen in other methods. If the primitives are first scan converted, then combined with CSG operations, errors may occur at the boundaries of the primitives, where exact surface information has been lost [20]. It is also possible to evaluate the CSG model to produce a polygonal approximation to the final object [14]. Payne and Toga's method may be used to then calculate a distance to the polygonal model. We preferred to make our calculations directly on the original model, and avoid the extra step of approximating the CSG model with polygons and the errors associated with calculating distance to a faceted model. Our approach also generates the additional closest point information, which may be used in a variety of graphics applications.

3 Generating the Distance Volume

This sections describes the two major components of our approach. The first step generates a set of closest points on the surface of the evaluated model. Additionally, it calculates the shortest distance to another set of points in a narrow band near the surface. The second step uses a Fast Marching Method to propagate this information to the remaining voxels of the distance volume.

3.1 Calculating Closest Points for the Narrow Band and Zero Set

The narrow band and zero set needed for the Fast Marching Method are generated with a modified version of the Constructive Cubes algorithm [3]. For each voxel, the algorithm involves traversing the CSG model's acyclic graph, evaluating each primitive's inside-outside function at the voxel location, and combining subcomponent values at each non-leaf node of the graph to produce a value which represents the inside-outside function for the complete model at a particular point. The original combination rules of Constructive Cubes are defined to produce a value of 1 for points on the surface of the evaluated CSG model. The Constructive Cubes algorithm was developed to calculate the final evaluated surface of a CSG model, which is produced by applying the Marching Cubes algorithm to the derived volume dataset with an iso-value of one. It was not developed to produce reasonable values away from the CSG model's surface.

The Constructive Cubes algorithm has been modified in two ways to generate the data needed for the fast marching algorithm. First, the step which evaluates the inside-outside function for each superellipsoid at a particular point has been replaced by a technique for calculating the closest point to the superellipsoid from an arbitrary point. Given the original evaluation point and the closest point on the superellipsoid, the shortest distance from the point to the superellipsoid can be calculated. The second modification involves formulating new shortest distance combination rules which are applied at each non-leaf node of the CSG graph. These are formulated in a way to produce the closest point and shortest distance to the entire CSG model from a point near the model surface.

3.1.1 Calculating the Closest Point to a Superellipsoid

The parametric equation for a superellipsoid is

$$\mathbf{S}(\eta,\omega) = \begin{bmatrix} a_1 cos^{\epsilon 1}(\eta) cos^{\epsilon 2}(\omega) \\ a_2 cos^{\epsilon 1}(\eta) sin^{\epsilon 2}(\omega) \\ a_3 sin^{\epsilon 1}(\eta) \end{bmatrix} \quad \begin{matrix} -\pi/2 \le \eta \le \pi/2 \\ -\pi \le \omega \le \pi \end{matrix} \qquad (1)$$

where η and ω are the longitudinal and latitudinal parameters of the surface, a_1, a_2, a_3 are the scaling factors in the x, y, and z directions, and $\epsilon 1$ and $\epsilon 2$ define the shape in the longitudinal and latitudinal directions [1].

The distance to the point on the surface of a superellipsoid defined at $[\eta,\omega]$ from an arbitrary point $\mathbf{P}$ is

$$d1(\eta,\omega) = ||\mathbf{S}(\eta,\omega) - \mathbf{P}||. \qquad (2)$$

Squaring and expanding Equation 2 gives

$$\begin{aligned} d2(\eta,\omega) \quad = \quad & (a_1 cos^{\epsilon 1}(\eta) cos^{\epsilon 2}(\omega) - P_x)^2 \\ & + (a_2 cos^{\epsilon 1}(\eta) sin^{\epsilon 2}(\omega) - P_y)^2 \\ & + (a_3 sin^{\epsilon 1}(\eta) - P_z)^2. \end{aligned} \qquad (3)$$

The closest point to the superellipsoid from an arbitrary point $\mathbf{P}$ can then be calculated by determining the values of $[\eta,\omega]$ which minimize Equation 3. In general Equation 3 is minimized with a gradient descent technique utilizing variable step-sizes. These values of $[\eta,\omega]$ may then be plugged into Equation 1 to give the closest point on the surface of the superellipsoid, which in turn may be used to calculate the shortest distance.

Several issues must be addressed when minimizing Equation 3. First, the special degenerate cases of the superellipsoid must be dealt with separately, because their surface normals are discontinuous. The most common cases are the cuboid ($\epsilon 1 = \epsilon 2 = 0$), the cylinder ($\epsilon 1 = 0, \epsilon 2 = 1$), the double cone ($\epsilon 1 = 2, \epsilon 2 = 1$), and the double pyramid ($\epsilon 1 = \epsilon 2 = 2$). The shortest distance to these primitives may be determined with non-iterative, closed form solutions.

Finding the values of η and ω at the closest point with a gradient descent technique involves calculating the gradient of Equation 3,

$$\nabla d2 = [\partial d2/\partial\eta, \partial d2/\partial\omega]. \qquad (4)$$

Unfortunately, superellipsoids have a tangent vector singularity near values of η or ω which are multiples of $\pi/2$. To overcome this problem, we reparameterize $\mathbf{S}$ by arc length [6]. That is,

$$\mathbf{S}(\eta,\omega) = \mathbf{S}(\eta(\alpha),\omega(\beta)) = \mathbf{S}(\alpha,\beta). \qquad (5)$$

where

$$\left\| \frac{\partial \mathbf{S}(\alpha,\beta)}{\partial\alpha} \right\| = 1 \text{ and } \left\| \frac{\partial \mathbf{S}(\alpha,\beta)}{\partial\beta} \right\| = 1. \qquad (6)$$

Given this we can say

$$\left\| \frac{\partial \mathbf{S}(\alpha,\beta)}{\partial\alpha} \right\| = \left\| \frac{\partial \mathbf{S}(\eta,\omega)}{\partial\eta} \right\| \cdot \left\| \frac{\partial\eta(\alpha)}{\partial\alpha} \right\| \qquad (7)$$

and

$$\left\| \frac{\partial \mathbf{S}(\alpha,\beta)}{\partial\beta} \right\| = \left\| \frac{\partial \mathbf{S}(\eta,\omega)}{\partial\omega} \right\| \cdot \left\| \frac{\partial\omega(\beta)}{\partial\beta} \right\|. \qquad (8)$$

If we assume that the arc-length parameterization is in the same direction as the original parameterization, we have

$$\frac{\partial\eta(\alpha)}{\partial\alpha} = \left\| \frac{\partial \mathbf{S}(\eta,\omega)}{\partial\eta} \right\|^{-1} \text{ and } \frac{\partial\omega(\beta)}{\partial\omega} = \left\| \frac{\partial \mathbf{S}(\eta,\omega)}{\partial\omega} \right\|^{-1}. \qquad (9)$$

Now we re-express our steepest descent (on d2) so that it is steepest with respect to the normalized parameters

$$\frac{\partial d2}{\partial\alpha} = \frac{\partial d2}{\partial\eta}\frac{\partial\eta}{\partial\alpha} = \frac{\partial d2}{\partial\eta}\left\| \frac{\partial \mathbf{S}(\eta,\omega)}{\partial\eta} \right\|^{-1} \qquad (10)$$

and

$$\frac{\partial d2}{\partial\beta} = \frac{\partial d2}{\partial\omega}\frac{\partial\omega}{\partial\beta} = \frac{\partial d2}{\partial\omega}\left\| \frac{\partial \mathbf{S}(\eta,\omega)}{\partial\omega} \right\|^{-1}. \qquad (11)$$

We now can use the gradient of the reparameterized $d2$,

$$\nabla d2' = [\partial d2/\partial\alpha, \partial d2/\partial\beta], \qquad (12)$$

to find the closest point with greater stability.

The general formulation of Equation 12 significantly simplifies for values of η and ω near multiples of $\pi/2$. Instead of deriving and implementing these simplifications for all regions of the superellipsoid we chose to only perform the calculation in the first octant ($0 \le \eta \le \pi/2, 0 \le \omega \le \pi/2$). Since a superellipsoid is 8-way symmetric, point $\mathbf{P}$ may be reflected into the first octant, the minimization performed, and the solution point reflected back into $\mathbf{P}$'s original octant.

3.1.2 Combining Shortest Distance Calculations

The CSG graph is processed in a depth-first manner. The closest point on and shortest distance to individual superellipsoids are calculated at the leaf nodes. The results from the non-leaf nodes' subcomponents (A and B) are then combined. Since the subcomponents may be combined with a variety of boolean operations (union, intersection and difference) just choosing the closest point to the subcomponents does not produce the correct result. Similar to CSG classification methods [19], a set of combination rules are utilized at each non-leaf node to evaluate the complete model, and are defined in Tables 1, 2, and 3. The rules are formulated for combining signed distance values which have no predefined limits. The values of A and B are negative inside an object and positive outside. Combination decisions are based on the signed distances computed from the non-leaf node's subcomponents. Additionally the closest point to the tested point is appropriately updated at each non-leaf node, until the complete model has been evaluated.

The entries in the tables have the following meanings. The IN conditions are used when the point being tested against subcomponent A or B is inside the subcomponent, and the shortest distance to that subcomponent is negative. The OUT conditions are used when the point being tested against subcomponent A or B is outside the subcomponent, and the shortest distance to that subcomponent is positive. The ON conditions are used when the point being tested against subcomponent A or B is on the subcomponent, and the shortest distance to that subcomponent is zero. MAX states that the two values may be combined by taking the maximum of the values

	A ∪ B	B IN	B OUT	B ON
	IN	MIN	A	A
A	OUT	B	MIN	B
	ON	B	A	A

Table 1: Union combination rules.

	A ∩ B	B IN	B OUT	B ON
	IN	MAX	B	B
A	OUT	A	MAX	A
	ON	A	B	A

Table 2: Intersection combination rules.

	A - B	B IN	B OUT	B ON
	IN	-B	MAX(A,-B)	B
A	OUT	MAX(A,-B)	A	A
	ON	-B	A	A

Table 3: Signed distance difference combination rules.

	A - B	B IN	B OUT	B ON
	IN	2-B	MAX(A,1/B)	B
A	OUT	MIN(A,2-B)	A	A
	ON	2-B	A	A

Table 4: Inside-outside difference combination rules.

returned by evaluating A and B. MIN states that the two values may be combined by taking the minimum of the two. 'A' states that the values of A and B are combined by taking the shortest distance to A. 'B' states that the values of A and B are combined by taking the shortest distance to B. '-B' states that the values of A and B are combined by taking the negative of B. MAX(A,-B) states that the combination is produced by taking the maximum of the value of A and the negative of B.

The combination rules for union and intersection are the same as the ones described in the original Constructive Cubes paper. A detailed explanation of these rules may be found in [3].

The combination rules for difference (A-B) have been changed to work with signed distances, and may be explained with Figure 1. Point P6 is the IN-IN condition. The shortest distance to the evaluated surface is the shortest distance to B. Since P6 is inside of B the shortest distance to B is negative. P6 is outside the evaluated model, and therefore must be negated to produce the correct signed distance. In the IN-OUT case, A is negative and B is positive. Therefore MAX(A,-B) compares two negative numbers, producing the number with the smallest absolute value. The correct answer for P1 is A, while the correct answer for P4 is -B. P5 is in A and on B. B or zero is the correct result for this combination. The OUT-IN combination rule is also MAX(A,-B). In this case A is positive and B is negative, and it compares two positive numbers, producing the distance with the largest absolute value. The correct answer at P7 is -B, recalling that B is negative, and must be negated to produce the correct signed result. The correct answer at P3 is A. P10 is the OUT-OUT condition, with A providing the closest point to the evaluated model. P12 is the OUT-ON condition, with A also being the correct answer. P8 represents the ON-IN condition. A is zero in this case, and B is negative. B is negated to produce the correct signed distance. P2 is the ON-OUT condition, which returns A, which is zero. The ON-ON case occurs at the intersection point of

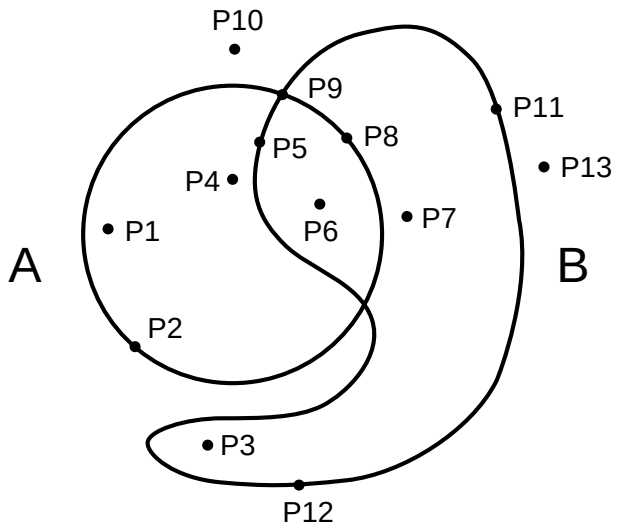


Figure 1: Evaluation points for a CSG model.

the two objects (P9), and returns A, which is zero.

It is not always possible to determine the closest point to a CSG model given the closest points to the primitives which comprise it. (From our experience this occurs in significantly less than one percent of the narrow band calculations.) As seen in Figure 1, no valid result can be calculated for P13, when evaluating A - B. Both of the closest points to A and B are not on the final evaluated model. Similarly, no valid solution can be generated at P6 when evaluating A ∪ B. The closest points to both A and B are on interior curves, which will not be a part of the final evaluated model. These invalid points which do not lie on the final evaluated CSG model can be easily removed from the zero set by evaluating them with the original Constructive Cubes algorithm. If the evaluation returns a value within a small ϵ of 1 (i.e., the point lies on the surface of the evaluated model), the point is retained. Otherwise it is discarded. Since the sampling of the zero set is quite dense, no adverse effects have been noted from discarding the occasional incorrect closest point. The first step of the algorithm produces a satisfactory distribution of closest points on the evaluated surface of the CSG model.

Even though the range of the superellipsoid's inside-outside $[0, \infty]$ is different than the signed distance $[-\infty, \infty]$, the inside-outside function combination rules for union and intersection used in the original Constructive Cubes algorithm are the same as the rules for combining signed distances, and are given in Tables 1 and 2. The inside-outside difference combination rules are different than the signed distance combination rules, and are given in Table 4.

3.2 Fast Marching Method For Computing Closest Points

We present a Fast Marching Method for computing the approximate closest point to a surface from the points in a regular grid. It is an approach based on the work of Sethian [16, 17]. His approach has been used in the past to numerically solve partial differential equations, but we have modified it to use a heuristic rule for propagating closest point information instead of calculating distance with a finite difference scheme. The accuracy of the method depends on a discretization of the surface and is independent of the volume grid spacing, allowing us to calculate distance to sub-voxel accuracy.

3.2.1 The Eikonal Equation and the Fast Marching Level Set Method

Let $u(x, y, z)$ denote the signed distance from the closed surface S. u satisfies the Eikonal equation,

$$|\nabla u| \equiv \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial u}{\partial z}\right)^2} = 1, \quad \text{subject to} \quad u\Big|_S = 0. \tag{13}$$

The characteristics of Equation 13 are straight lines that are normal to S and point in the direction of increasing distance. For each point (x, y, z) in space, there is a line segment from the surface to the point that is a characteristic of the entropy-satisfying solution of the Eikonal equation. The point (x, y, z) and the closest point on the surface S are the endpoints of this line segment.

Sethian [16, 17] has developed a Fast Marching Level Set Method to solve the Eikonal equation,

$$|\nabla u| f(x, y, z) = 1, \quad \text{subject to} \quad u\Big|_S = g(x, y, z),$$

in the case that f is either always positive or negative. The method uses an upwind, viscosity solution, finite difference scheme to numerically solve this equation. For $f(x, y, z) = 1$ and $g(x, y, z) = 0$, the solution gives the signed distance from the surface S. The initial condition $u\Big|_S = 0$ is specified by giving the value of u on a narrow band of points around the surface S. The distance values in the remainder of the volume are computed by pushing this narrow band outward.

3.2.2 Closest Point Calculation Overview

To calculate the closest points to a surface on a regular grid, we utilize Sethian's Fast Marching Method, but instead of using a finite difference scheme to compute distance, we use a heuristic algorithm to propagate closest points information. Instead of specifying the distance for the points in the narrow band as an initial condition, we specify the closest points to the surface. In one step of the closest points method:

1. The point gp with the smallest distance is removed from the narrow band and it's value is frozen.
2. Points are added to the narrow band to maintain unit thickness.
3. The closest points of the neighbors with larger distances than gp are recomputed using the closest point information from gp.

The closest points method is based on the following idea. The closest point on the surface to a point in the grid is usually close to one of the closest points of its neighbors in the grid. Thus if one knows the closest points of the neighbors of a grid point gp, one can compute an approximate closest point for gp by assuming that it is near one of the closest points of its neighbors. This is only a heuristic, and in Figure 2 we see cases in two dimensions for which the heuristic succeeds and fails. In the cases where the heuristic fails to determine the correct closest point, it still gives a reasonable approximation of the distance. The heuristic may fail if the characteristics from several different portions of the surface S intersect near gp. Fortunately, if the heuristic fails at a point, this mistake is usually not propagated outward to increasing distances. This is because "information" in the Eikonal equation and the closest points method is propagated along characteristics of Equation 13. Where characteristics collide, information goes into the shock and is lost.

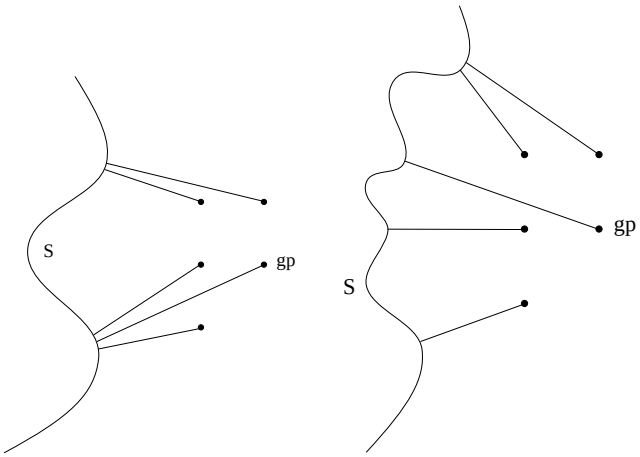


Figure 2: The Closest Point Heuristic.

3.2.3 Terminology

Let the *distance volume* be the $N \times N \times N$ grid that spans the space around the scan converted object. We will refer to points in the distance volume with (i, j, k) coordinates. Let the *zero set grid* be an $M \times M \times M$ uniform grid that spans the same Cartesian domain. We refer to the ratio M/N as the *super-sampling factor* of the zero set grid. In most cases the zero set grid is finer than the distance volume grid, providing distance calculations with sub-voxel accuracy. We will refer to points in the zero set grid with (I, J, K) coordinates. For any grid point, the closest point is defined as the Cartesian coordinates of the point on the CSG model surface that is closest to that grid point.

3.2.4 Initial Data

The fast marching algorithm takes as input: a set of points in the distance volume that forms a *narrow band* around the CSG model surface and a point sampling of the surface. The narrow band contains all the points in the distance volume having the property that a neighbor of the point has opposite inside/outside status.[1] We generate the narrow band by evaluating the inside/outside status [19] of all the grid points of the distance volume, and note where inside/outside transitions occur. For the points in the narrow band we must supply the (i, j, k) coordinates of the points and their inside/outside status. The narrow band is used as a starting point for propagating the closest point information outward and inward to the rest of the points in the distance volume. Note that specifying the inside/outside status of the points in the narrow band determines the inside/outside status of the other points in the grid.

During this stage of our calculations the CSG model surface is represented with a set of points that lie on the surface. This set of points will be called the *zero set*, as they are points lying on the isosurface of zero distance. The zero set is made by first constructing a thin band of points in the zero set grid that surrounds the CSG model surface. This set of grid points will be called the *zero band*. The zero set is the set of closest points on the model surface to the grid points in the zero band. The method used to calculate the zero set has been described in the previous section. Given a point p in the zero set that is closest to the grid point (i, j, k) in the zero band, one can determine all the points in the zero set that lie in

[1] In three dimensions *neighbor* means one of the 26 locations surrounding each grid point.

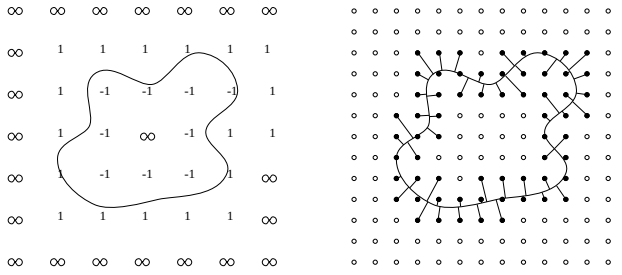


Figure 3: Initial Data for the Fast Marching Algorithm.

a neighborhood of p by determining all the points in the zero band in a neighborhood of (i, j, k). As input to the algorithm, we must supply the (I, J, K) coordinates of the grid points in the zero band and the corresponding (x, y, z) coordinates of the points in the zero set. In Figure 3 the initial data is shown pictorially in 2 dimensions.

3.2.5 Propagating the Closest Point Data

Initially, we have the closest points data in the zero band that surrounds the surface. We use the closest points data in the zero band to determine the closest points in the narrow band of the distance volume and then march the narrow band outward and inward to calculate the closest points in the rest of the distance volume. Consider a point gp that neighbors the band and whose closest point is unknown. The closest point of gp is probably close to one of the closest points of its neighbors in the band. Thus for each neighbor of gp in the band, we recompute the distance of gp by considering points that are near the closest points of that neighbor. First we will present the marching algorithm that moves the band outward and then inward. Next, we will show the algorithm for recomputing the distance at a point gp, given the closest point of one of its neighbors.

Let in_out_{ijk} denote the inside/outside status for a point in the distance volume; $+1$ for outside, -1 for inside. Let grid_{ijk} denote the computed distance at a distance volume grid point. A value of ∞ indicates that the distance has not yet been computed. Let source_{ijk} denote the point in the zero set Z from which this distance was computed.

Initially: The closest point to each (I, J, K) in the zero band is known. For each (i, j, k) in the narrow band $\text{grid}_{ijk} = \text{in\_out}_{ijk}$. For each point not in the narrow band grid_{ijk} and in_out_{ijk} are set to be undefined. The closest points of the zero band are used to generate approximate closest points for the narrow band. Below is the fast marching, closest points algorithm.

begin
```
// March forward to find positive distances.
put each point with a non-negative, finite
 grid_ijk in the set U;
while U ≠ ∅
   remove the grid point gp with the smallest
    distance from U;
   for each of the 26 neighbors of gp
      if the source of the neighbor is unknown
         add that neighbor to U;
      if the distance of the neighbor is
       larger than the distance of gp
         recompute the neighbor's distance
          using gp's source s;
```
end

Next the narrow band is marched backward to compute the closest points with negative distance. Below is the algorithm to recompute the distance grid_{ijk} to the distance grid point gp, using a zero set source s. Let (I, J, K) be the coordinates in the zero band for which s is the closest point. The user chooses the search radius parameter R. This is the radius of a cube around the point (I, J, K) in the zero band that defines a neighborhood on the surface around the point s. The parameter, $\sigma = 2 * R + 1$ is the diameter of the cube. When recomputing the distance, all the points in the zero set in a neighborhood around s are considered as possible closest points.

begin
```
for each grid point (l, m, n) in a σ × σ × σ cube
 surrounding (I, J, K)
   t ∈ Z is the closest point to (l, m, n);
   calculate the distance from gp to t;
grid_ijk = minimum of the σ^3 computed distances;
source_ijk = the source of this minimum distance,
 (an element of Z);
```
end

From experience we have found that for most surfaces, a search radius R of half the super-sampling factor of the zero set grid will provide satisfactory closest points information to the set Z. Finally, note that since the zero band is of small constant thickness, the number of points in the zero band in the $\sigma \times \sigma \times \sigma$ cube is $\mathcal{O}(\sigma^2)$.

3.3 Computational Complexity

There are N^3 grid points in the distance volume. Each distance grid point is removed from the narrow band once, giving us a factor of N^3. At any point in the algorithm, there are $\mathcal{O}(N^2)$ points in the narrow band. There are $2P$ nodes in the binary tree representing the CSG model, where P is the number of superellipsoids in the model. Each node of the model must be evaluated (in constant time) to determine if a particular grid point is inside or outside the model. Determining which grid points are in the initial narrow band requires $\mathcal{O}(N^3 P)$ operations. Determining the closest point on the CSG model from a particular grid point is also an $\mathcal{O}(P)$ operation. This is only computed on the points of the zero band. Calculating the zero set requires $\mathcal{O}(M^2 P)$ operations. Unfortunately it is difficult to characterize the amount of time needed to calculate the closest point to each superellipsoid, since each one is evaluated with an iterative technique. This calculation typically requires approximately 30 iterations in our variable step-size gradient descent routine.

The cost of adding and deleting elements from the narrow band is proportional to the logarithm of the number of points in the narrow band. This gives us a factor of $\mathcal{O}(\log N)$. The computational cost of recomputing the distance for a given grid point is proportional to the number of zero band points in a $\sigma \times \sigma \times \sigma$ cube neighborhood of a point s in the zero band. This gives us a factor of $\mathcal{O}(\sigma^2)$. Thus the overall computational complexity of the fast marching algorithm is $\mathcal{O}(N^3 \sigma^2 \log N)$.

4 Results

A number of moderately complex CSG models have been scan converted into distance volumes with our approach. Each of the CSG models consist of superellipsoids which have been unioned, intersected, and/or differenced to produce the final shapes. The resulting distance volumes have been used to generate an evaluated surface of the model, as well as offset surfaces. Additionally, the volumes have been utilized to morph one model into another [21]. Figure 7 presents an improved evaluated surface of a CSG model

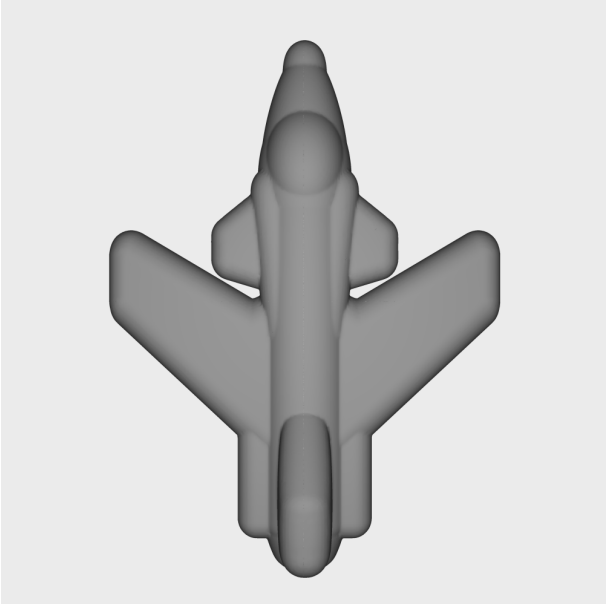


Figure 4: Offset surface from the X-29 distance volume.

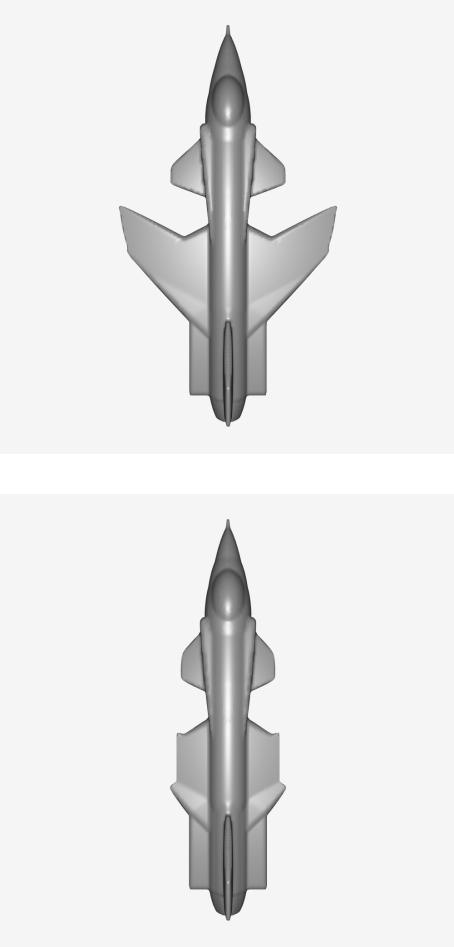


Figure 5: A 3-D morphing between an X-29 and a dart.

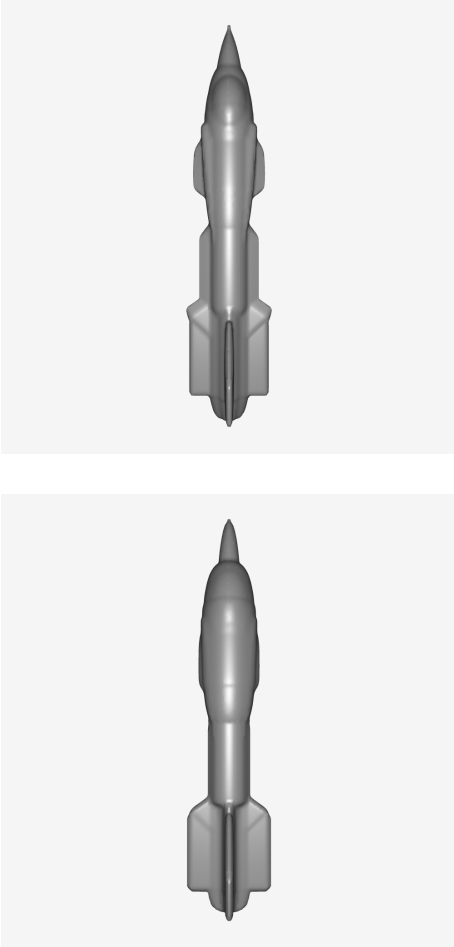


Figure 5: A 3-D morphing between an X-29 and a dart. (cont.)

similar to the one included in the first Constructive Cubes paper. The polygonal model is generated by applying the Marching Cubes algorithm [12] to a distance volume of dimension $195 \times 90 \times 120$, producing an iso-surface of value zero. The result presented here is superior to the one presented in the original paper. See Figure 6. Since the Marching Cubes algorithm linearly interpolates the iso-surface value between volume grid points (voxels), utilizing shortest Euclidean distance instead of the non-linear superellipsoid inside-outside functions values provides the linear relationship necessary for correctly calculating the iso-surface intersection point between each voxel in the Marching Cubes algorithm, and for properly combining subcomponent values in the Constructive Cubes algorithm.

Figure 8 presents the zero iso-surface of a model of an X-29 jet fighter, consisting of 38 primitives and also generated from a $96 \times 192 \times 240$ distance volume. Figure 9 presents the iso-surface of value zero of a dart model, consisting of 21 primitives, generated from a $96 \times 192 \times 240$ distance volume. These volume resolutions were chosen because they produced satisfactory results given the cost in time (several hours) and memory ($\sim$17 MBtyes) to produce them. The excessive time needed to produce our results is significantly affected by the message-passing overhead imposed by the object-oriented environment used to prototype our algorithms [8]. We believe that the processing times can be improved by at least an order magnitude if the algorithm is custom coded in a conventional programming environment. Figure 4 presents an offset surface to the X-29 model. This is the iso-surface at value 0.5

running through the X-29's distance volume. The colors in Figures 7, 8 and 9 were generated with the closest point information that is maintained with the shortest distance information. Discussion of this aspect of our work is beyond the scope of this paper. Figure 5 presents four intermediate shapes produced while morphing the X-29 model in Figure 8 into the dart model in Figure 9. The X-29 model follows the dart's shortest distance information to the surface of the dart [21].

5 Conclusion

We have described a technique for generating a distance volume with sub-voxel accuracy from one type of geometric model, a CSG model consisting of superellipsoid primitives. The distance volume is generated in a two step process. The first step calculates the shortest distance to the CSG model at a set of points within a narrow band around the evaluated surface. Additionally, a second set of points, labeled the zero set, which lies on the CSG model's surface are computed. A point in the zero set is associated with each point in the narrow band. Once the narrow band and zero set are calculated, a Fast Marching Method is employed to propagate the shortest distance and closest point information out to the remaining voxels in the volume. Our technique has been used to scan convert a number of CSG models, producing distance volumes which have been utilized in a variety of computer graphics applications, e.g. CSG surface evaluation, offset surface generation, and 3-D model morphing.

6 Acknowledgements

We would like to thank Dr. Alan Barr and the other members of the Caltech Computer Graphics Group for their support and assistance. Timothy Doyle created the model used in Figure 9. This work was financially supported by the National Science Foundation (ASC-89-20219), as part of the STC for Computer Graphics and Scientific Visualization; the National Institute on Drug Abuse, the National Institute of Mental Health and the NSF, as part of the Human Brain Project; the Office of the Director of Defense Research and Engineering, and the Air Force Office of Scientific Research (F49620-96-1-0471), as part of the MURI program; and the Volume Visualization Program of the Office of Naval Research (N00014-97-0227). Additional equipment grants were provided by Silicon Graphics, Hewlett-Packard, IBM, and Digital Equipment Corporation. This work was initially funded by the former shareholders of the European Computer-Industry Research Centre: Bull SA, ICL PLC, and Siemens AG.

REFERENCES

[1] A. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981.

[2] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34:344–371, 1986.

[3] D. E. Breen. Constructive Cubes: CSG evaluation for display using discrete 3D scalar data sets. In Werner Purgathofer, editor, *Eurographics '91*, pages 127–142. North-Holland, September 1991.

[4] D. Cohen and A. Kaufman. Scan-conversion algorithms for linear and quadratic objects. In A. Kaufman, editor, *Volume Visualization*, pages 280–301. IEEE Computer Society Press, 1990.

[5] D. Cohen-Or, D. Levin, and A. Solomivici. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, 1998.

[6] M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, Englewood Cliffs, NJ, 1976.

[7] S. Fang and R. Srinivasan. Volumetric-CSG – a model-based volume visualization approach. In *Proceedings of the 6th International Conference in Central Europe on Computer Graphics and Visualization*, 1998.

[8] P. Getto and D. Breen. An object-oriented architecture for a computer animation system. *The Visual Computer*, 6(2):79–92, March 1990.

[9] M.W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, 1996.

[10] A. Kaufman. An algorithm for 3D scan-conversion of polygons. In G. Marechal, editor, *Eurographics '87*, pages 197–208. North-Holland, August 1987.

[11] A. Kaufman. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 171–179, July 1987.

[12] W.E. Lorensen and H.E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[13] B. Payne and A. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, 12(1):65–71, 1992.

[14] A. Requicha and H. Voelcker. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(1):30–44, 1985.

[15] A. A. G. Requicha and H. B. Voelcker. Solid modeling: a historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2):9–22, March 1982.

[16] J.A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Science*, volume 93 of 4, pages 1591–1595, 1996.

[17] J.A. Sethian. *Level Set Methods*. Cambridge University Press, Cambridge, UK, 1996.

[18] N. Shareef and R. Yagel. Rapid previewing via volume-based solid modeling. In *Proceedings of the 3rd Symposium on Solid Modeling and Applications*, pages 281–292, May 1995.

[19] R. B. Tilove. Set membership classification: a unified approach to geometric intersection problems. *IEEE Trans. Comput.*, C-29:874–883, October 1980.

[20] S. Wang and A. Kaufman. Volume-sampled 3D modeling. *IEEE Computer Graphics and Applications*, 14(5):26–32, September 1994.

[21] R. Whitaker and D. Breen. Level-set models for the deformation of solid objects. In *Proceedings of the Third International Workshop on Implicit Surfaces*, pages 19–35. Eurographics Association, June 1998.

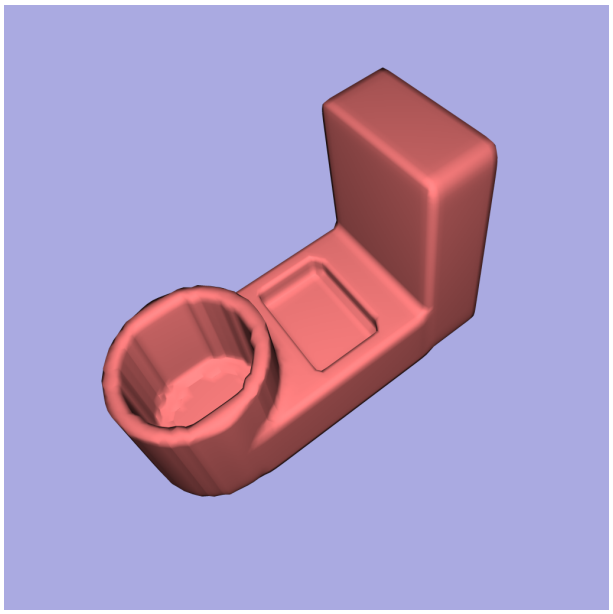


Figure 6: CSG model from the original Constructive Cubes paper.

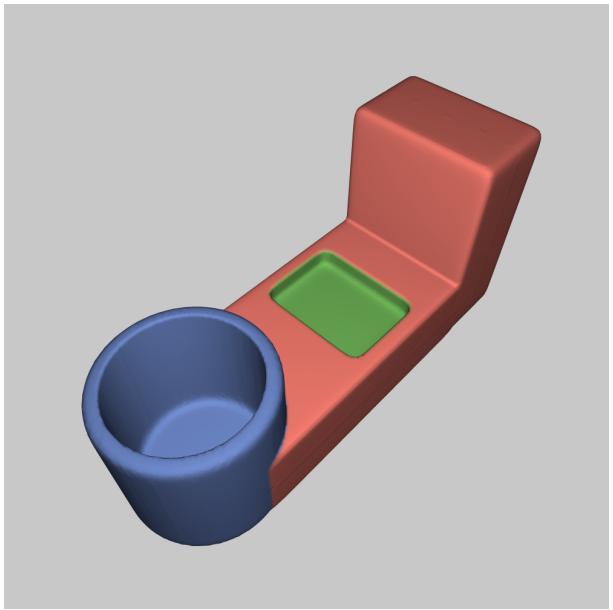


Figure 7: Improved surface evaluation utilizing distance volumes on a similar model.

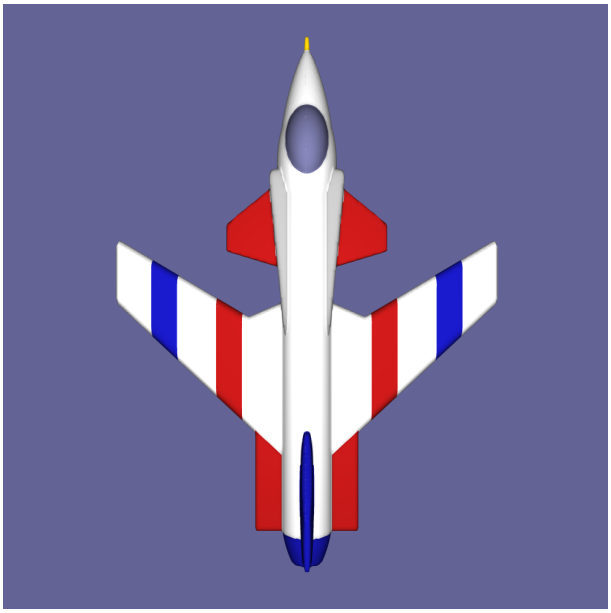


Figure 8: X-29 CSG model surface evaluation utilizing distance volumes.

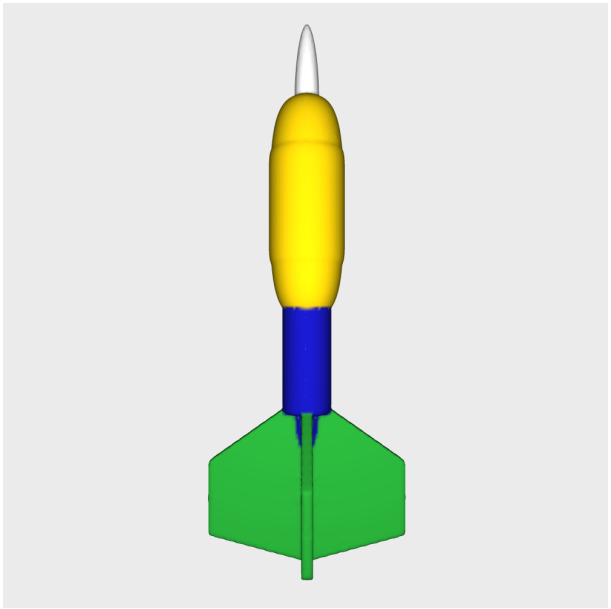


Figure 9: Dart CSG model surface evaluation utilizing distance volumes.

A Fast Algorithm for Computing the Closest Point and Distance Transform

Sean Mauch

December 4, 2000

Abstract

This paper presents a new algorithm for computing the *closest point transform* to a manifold on a rectilinear grid in low dimensional spaces. The closest point transform finds the closest point on a manifold and the Euclidean distance to a manifold for all the points in a grid, (or the grid points within a specified distance of the manifold). We consider manifolds composed of simple geometric shapes, such as, a set of points, piecewise linear curves or triangle meshes. The algorithm computes the closest point on and distance to the manifold by solving the Eikonal equation $|\nabla u| = 1$ by the method of characteristics. The method of characteristics is implemented efficiently with the aid of computational geometry and polygon/polyhedron scan conversion. The computed distance is accurate to within machine precision. The computational complexity of the algorithm is linear in both the number of grid points and the complexity of the manifold. Thus it has optimal computational complexity. Examples are presented for piecewise linear curves in 2D and triangle meshes in 3D.

1 The Closest Point Transform

Let $u(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$, be the distance from the point $\mathbf{x}$ to a manifold S. If $\dim(S) = n - 1$, (for example curves in 2D or surfaces in 3D), then the distance is signed. The orientation of the manifold determines the sign of the distance. One can adopt the convention that the outward normals point in the direction of positive or negative distance. In order for the distance to be well-defined, the manifold must be orientable and have a consistent orientation. A Klein bottle in 3D for example is not orientable. Two concentric circles in 2D have consistent orientations only if the normals of the inner circle point "inward" and the normals of the outer circle point "outward", or vice-versa. Otherwise the distance would be ill-defined in the region between the circles. For manifolds which are not closed, the distance is ill-defined in any neighborhood of the boundary. However, the distance is well-defined in neighborhoods of the manifold which do not contain the boundary. If $\dim(S) < n - 1$, (for example a set of points in 2D or a curve in 3D), the distance is unsigned, (non-negative).

The distance can also be shown to be the entropy-satisfying solution or vanishing viscosity solution of an Eikonal equation. Consider a surface S that moves in a direction normal to itself with speed f. Let $u(\mathbf{x})$ be the arrival time of the surface at the point $\mathbf{x}$. $|\nabla u|$ has magnitude $1/f$. On the surface S, u is zero. Thus the arrival time is the solution of

$$|\nabla u| = \frac{1}{f}, \quad u\big|_S = 0.$$

If the speed is unity, then the arrival time is the distance from the surface. The distance from the surface satisfies the Eikonal equation,

$$|\nabla u| = 1, \quad u\big|_S = 0. \tag{1}$$

The solution is C^0.

Let $\boldsymbol{\xi}$ be the closest point on a manifold to the point $\mathbf{x}$. The distance to the manifold is $|\mathbf{x} - \boldsymbol{\xi}|$. $\mathbf{x}$ and $\boldsymbol{\xi}$ are the endpoints of the line segment that is a characteristic of the solution of Equation 1. If the manifold is smooth then the line connecting $\mathbf{x}$ to $\boldsymbol{\xi}$ is orthogonal to the manifold. If the manifold is not smooth at $\boldsymbol{\xi}$ then the line lies "between" the normals of the smooth parts of the manifold surrounding $\boldsymbol{\xi}$.

The distance transform is the value of the distance for the points in a grid that surrounds the surface in question. It transforms an explicit representation of a manifold into an implicit one. The manifold is implicitly represented as the level set of distance zero. The inverse operation, converting the distance volume data to a manifold may be accomplished with algorithms such as Marching Cubes [3]. The closest point transform is the value of the closest point on the manifold for the points in the grid.

The distance and closest point transforms are important in several applications which we discuss briefly below. The distance transform can be used to convert an explicit surface into a level set representation of the surface. The surface is the isosurface of value zero. Algorithms for working with the level set are often simpler and more robust than dealing with the surface directly. Set operations like union and intersection for faceted surfaces or polynomial patch surfaces are difficult to implement. However, set operations on the level set representation of the surfaces are trivial.

Another example of the utility of the distance transform is the application to surface propagation problems in which a surface moves with a given normal velocity. A simple example is surface offsetting, finding the surface that is a given distance from another surface. Working with the surface explicitly can be difficult because one needs ad hoc methods for dealing with self-intersections and topological changes. By working with the level set representation of the surface one can describe these problems with partial differential equations and use finite difference methods for their solution [5]. Surface offsetting is trivial when the level set representation of a surface

is the distance transform. The surface a distance d from the given surface is just the iso-surface of value d.

The closest point transform is useful when one needs information about the closest point on a surface in addition to the distance. Each point on a surface has a position and may have an associated velocity, color, or other data. One can use the closest point transform to do surface offsetting with color.

The closest point transform is useful in certain coupled solid mechanics / fluid mechanics computations in which we want to explicitly track the location of the solid-fluid interface [4]. Using a closest point transform, a Lagrangian solid mechanics code can communicate the position and velocity of the solid interface to an Eulerian fluid mechanics code. Consider a fluid grid which spans the entire domain, inside and outside the solid. Thus only a portion of the grid points lie in the fluid. Suppose further that the solid mechanics is done on a tetrahedral mesh. The boundary of the solid is a triangle mesh surface. Computing the distance transform to this surface on the fluid mechanics grid indicates which grid points are outside the solid and thus in the fluid domain. Through the closest point transform one can implement boundary conditions for the fluid at the solid boundary. In particular, it is necessary to recreate the closest point transform at each time step if the solid / fluid interface is itself time dependent. For such simulations it is highly desirable that the closest point transform have linear computational complexity in both the size of the fluid grid and solid mesh. If the closest point transform (CPT) does not have linear computational complexity, determining the fluid boundary condition through the CPT would likely dominate the computation.

The algorithm developed in this paper computes distance and closest point to manifolds which are composed of simple geometric primitives. The manifold may be given as sets of points, curves composed of line segments or surfaces composed of triangle facets. As an example, Figure 1 shows a triangular mesh. The second picture is a density plot of a slice of the distance. The final picture shows the closest point transform calculated for the grid points close to the surface. The closest point is depicted as line segments from grid points to closest points on the surface.

Before presenting the details of this new algorithm we will examine previous work on the distance and closest point transform. We will consider the geometrically based methods for computing the closest point transform and the finite difference based methods for computing approximate distance transforms. Then we will cover some background material that is prerequisite for developing an improved closest point transform algorithm. This algorithm will be demonstrated first in the context of computing the closest point transform to a piecewise linear curve in 2D and then a triangle mesh surface in 3D. Finally, we will examine the performance of the improved closest point transform algorithm and compare its performance to some other methods.

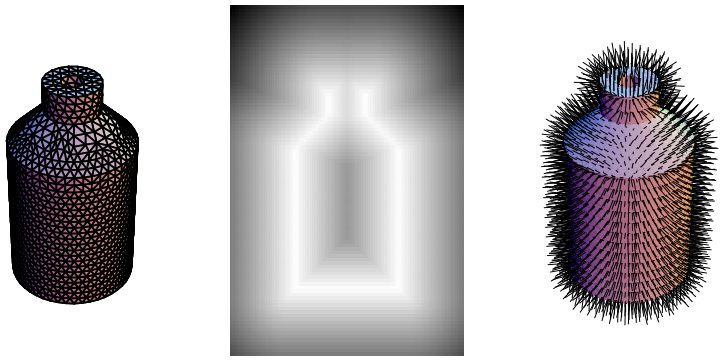


Figure 1: A Triangle Mesh, its Distance Transform and its Closest Point Transform

2 Previous Work

Brute Force Approach. The closest point transform to a manifold may be computed directly by iterating over the geometric primitives in the manifold as one iterates over the grid points. Consider a manifold S composed of M geometric primitives, p_m. Let d_{ijk} and cp_{ijk} denote the distance to the manifold and the closest point on the manifold for the points in a 3D grid of dimensions $n_x \times n_y \times n_z$. The brute force algorithm for computing the distance and closest point transform is:

initially
$\{\ d_{ijk} = \infty$ *: for all* $i, j, k\ \}$
begin
// Loop over the geometric primitives.
for $m \in [0..M)$*:*
    // Loop over the grid points.
    for $i \in [0..n_x)$*:*
        for $j \in [0..n_y)$*:*
            for $k \in [0..n_z)$*:*
                $d_{\text{new}} =$ *distance to* p_m
                if $|d_{\text{new}}| < |d_{ijk}|$*:*
                    $d_{ijk} = d_{\text{new}}$
                    $cp_{ijk} =$ *closest point on* p_m
            end // for
        end // for
    end // for
end // for
end

Since there are M geometric primitives in the manifold and $N = n_x \times n_y \times n_z$ grid points, the computational complexity of the brute force algorithm is $\mathcal{O}(MN)$. If the distance and closest point are only needed in a neighborhood around the manifold,

then the three inner *for* loops are replaced by a loop over all the grid points within a certain distance of the geometric primitive p_m.

The brute force algorithm is slow. However, it is embarrassingly concurrent with respect to both the distribution of the geometric primitives and the grid points. If the grid points are distributed over a number of processors, then the concurrent algorithm consists of each processor executing the above sequential algorithm on its share of the grid points. Next assume that the geometric primitives are distributed over a number of processors with each processor holding the entire grid. After each processor executes the sequential code with its share of the primitives, the grids can be merged by choosing the smallest distance for each of the grid points.

Finite Difference Methods. One can use upwind finite difference methods to solve the Eikonal equation, (Equation 1), and obtain an approximate distance transform [5]. The initial data is the value of the distance on the grid points surrounding the surface. This initial condition can be generated with the brute force method. An upwind finite difference scheme is then used to propagate the distance to the rest of the grid points. The scheme may be solved iteratively, yielding a computational complexity of $\mathcal{O}(\alpha N)$, where α is the number of iterations required for convergence. The scheme may also be solved by ordering the grid points so that information is always propagated in the direction of increasing distance. This is Sethian's fast marching method [6]. The computational complexity is $\mathcal{O}(N \log N)$.

There has been recent work in computing the distance transform with geometrically based finite difference schemes [7]. The solution is propagated outward from the initial surface by sweeping through the grid in each diagonal coordinate direction. The computational complexity of the algorithm is $\mathcal{O}(\alpha 2^d N)$, where d is the dimension and α is the number of iterations. The number of iterations required for this method is small. By using sophisticated finite difference schemes one can compute the distance accurately. If the manifold is a set of points, the distance can be computed to within machine precision.

LUB-Tree Methods. One can also use lower-upper-bound tree methods to compute the distance and closest point transforms [2]. The surface is stored in a tree data structure in which each subtree can return upper and lower bounds on the distance to any given point. This is accomplished by constructing bounding boxes around each subtree. For each grid point, the tree is searched to find the closest point on the surface. As the search progresses, the tree is pruned by using upper and lower bounds on the distance. Since the average computational complexity of each search is $\mathcal{O}(\log M)$, the overall complexity is $\mathcal{O}(N \log M)$.

3 Background Material

In order to develop the present algorithm for computing the closest point transform, we introduce the concepts of scan conversion and Voronoi diagrams.

Scan conversion or *rasterization* is a standard technique in computer graphics for displaying filled polygons on raster displays. It is a method for determining the pixels on the display which lie inside a polygon. Consider a convex polygon and a rectilinear grid. We can use scan conversion to efficiently determine the grid points which lie inside the polygon (see Figure 2). For each grid row that intersects the polygon we find the left and right intersection points and mark each grid point in between as being inside the polygon. Let e be the number of edges of the polygon, let r be the number of rows that intersect the polygon and let n be the number of grid points inside it. The computational complexity of the scan conversion algorithm is $\mathcal{O}(e+r+n)$. If the sides of the polygon are not smaller than the grid spacing, then the computational complexity is $\mathcal{O}(n)$.

Now consider a convex polyhedron and a 3D rectilinear grid. We can scan convert the polyhedron by intersecting the polyhedron with the planes that coincide with the grid rows to form polygons. This reduces the problem to polygon scan conversion. Figure 2 shows a pyramid and the polygons formed by intersecting with grid rows. If the sides of the polyhedron are not smaller than the grid spacing, then the computational complexity is linear in the number of grid points inside the polyhedron.

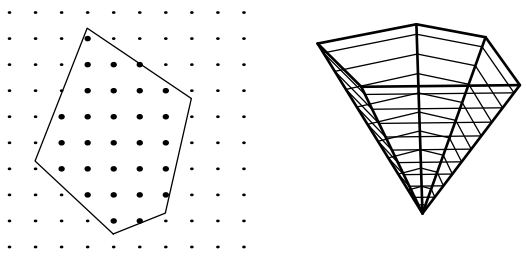


Figure 2: Scan Conversion of a Polygon in 2D. Slicing a Polyhedron to Form Polygons.

Consider a set of points $P = \{p_1, p_2, \ldots, p_M\}$ in $\mathbb{R}^2$. The *Voronoi diagram* [1] is a subdivision of the plane into M cells such that the *Voronoi cell* corresponding to p_i, $V(p_i)$, contains all the points in $\mathbb{R}^2$ to which p_i is the closest point in P. Each Voronoi cell is a bounded or unbounded convex polygon. If we consider a set of points P in the rectangle $[x_0, x_1] \times [y_0, y_1] \subset \mathbb{R}^2$ then each Voronoi cell is a bounded convex polygon. (See Figure 3.) For a set of points in $\mathbb{R}^3$ the Voronoi cells are convex polyhedra. The computational complexity of computing the Voronoi diagram is $\mathcal{O}(M \log M)$ in 2D, and $\mathcal{O}(M^2)$ in 3D.

As a step toward computing the closest point transform to a curve, we consider the closest point transform to a set of points $P = \{p_1, \ldots, p_M\}$. For each point in a rectilinear grid we will compute the closest point in P. We proceed by first finding the Voronoi diagram of P. For each Voronoi cell, $V(p_i)$, we use scan conversion to determine the grid points which lie inside this Voronoi polygon/polyhedron. The distance to p_i is computed for each of these grid points and p_i is the closest point.

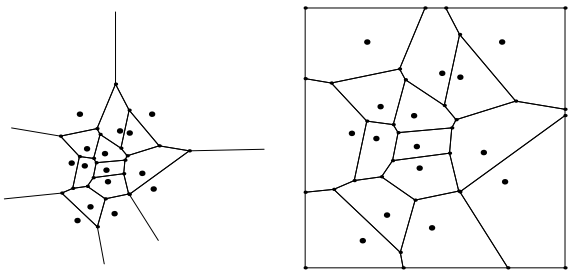


Figure 3: Unbounded and Bounded Voronoi Diagrams.

In implementing the algorithm, the polygons/polyhedra must be enlarged slightly to make sure that grid points are not missed due to finite precision arithmetic. As a byproduct of enlarging the polygons/polyhedra, some grid points may be scan converted more than once. In this case, the smaller distance and thus the closer point is chosen.

If the edges of the Voronoi polygons/polyhedra are no smaller than the grid spacing, then the computational complexity of the scan conversion will be linear in the number of interior grid points. In this case, the computation complexity of the closest point transform is $\mathcal{O}(M \log M + N)$ in 2D and $\mathcal{O}(M^2 + N)$ in 3D, where M is the number of points in P and where N is the number of grid points.

4 An Improved CPT Algorithm

In this section we develop an improved closest point transform algorithm for computing the closest point to a manifold for the points in a regular grid. We will use an similar to that for computing the CPT to a set of points. As a first step in the algorithm, we need something like a Voronoi diagram for the manifold. Instead of computing polygons/polyhedra that exactly contain the closest grid points to a point, we will compute polygons/polyhedra that at least contain the closest grid points to the components of the manifold. These polygons/polyhedra can then be scan converted to determine the grid points that are possibly closest to a given component.

4.1 The CPT for Piecewise Linear Curves

Consider the distance to a piecewise linear curve. For a given point, the closest point on a piecewise linear curve either lies on one of the edges or at one of the vertices. Suppose that the closest point on the curve $\boldsymbol{\xi}$ to a given point $\mathbf{x}$ lies on an edge. The vector from $\boldsymbol{\xi}$ to $\mathbf{x}$ is orthogonal to the line segment. Thus the closest points to a given line segment must lie within an infinite strip. The strip defined by the line segment and the (outward/inward) normals contains the points of (positive/negative) distance from the line. See Figure 4 for an illustration of the positive and negative strips for a simple curve. Note that the strips for each edge exactly contain the characteristic curves of the Eikonal equation emanating from that edge.

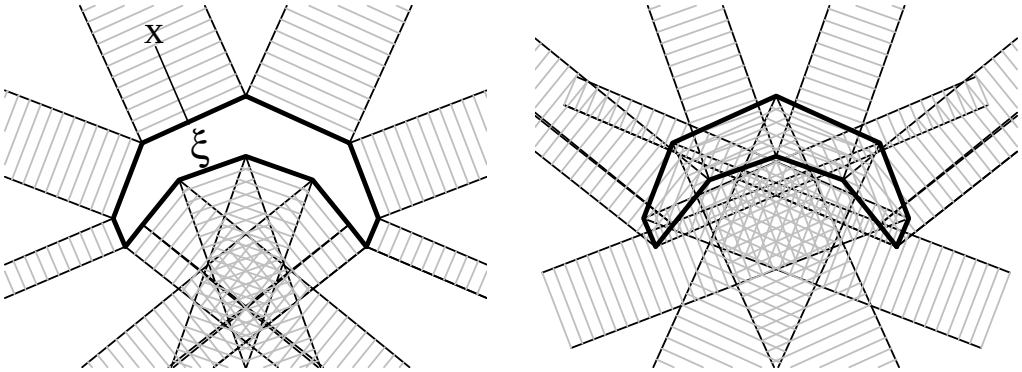


Figure 4: Strips Containing Points with Positive / Negative Distance to Edges.

Next, consider a point $\mathbf{x}$ whose closest point $\boldsymbol{\xi}$ is at a vertex. The vector from $\boldsymbol{\xi}$ to $\mathbf{x}$ must lie between the normal vectors to the two adjacent line segments at the vertex. Thus the closest points to a vertex must lie in a wedge. If the (outside/inside) angle between two adjacent line segments is less than π, then there are no points of (positive/negative) distance from the vertex. See Figure 5 for an illustration of the positive and negative wedges. These wedges again exactly contain the characteristic curves of the Eikonal equation emanating from the vertices.

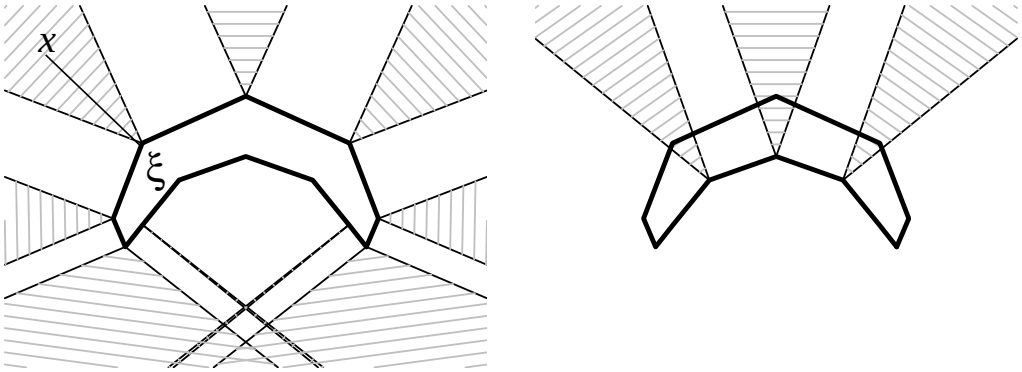


Figure 5: Wedges Containing Points with Positive / Negative Distance to Vertices

Now we consider computing the distance and closest point transform to a distance d away from the curve. We use the fact that the closest points to edges/vertices lie in strips/wedges to construct polygons which contain the points within a distance d. (See Figure 6.) For vertices, these are wedge-shaped polygons containing points of either positive or negative distance. For edges, these are rectangles containing points of both positive and negative distance. Note that these polygons are similar to Voronoi cells. They contain at least, (instead of exactly), the points which are closest to the edge or vertex. By using scan conversion, we can determine the grid

points which lie inside each polygon. We can use simple formulas from geometry to compute the distance and closest point for a given line segment or vertex.

If the closest point transform is being computed to a relatively large distance away from the curve, the polygons which contain the closest points may have large overlaps. In this case, we can reduce the size of these polygons by using information about the curve to clip them. This will result in fewer scan converted grid points and hence fewer closest point calculations. The details of this optimization are omitted.

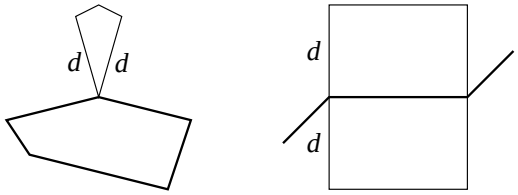


Figure 6: Polygons Containing Points within a Distance d of a Vertex and an Edge

We can now describe a fast algorithm for computing the distance and closest point transform to a piecewise linear curve. Let E be the set of edges and V the set of vertices. Let d_{ij} and cp_{ij} denote the distance to the curve and the closest point on the curve for the points in a 2D grid.

initially
{ $d_{ij} = \infty$: *for all* i, j }
begin
// Loop over the edges.
for each $e \in E$*:*
    p =*polygon containing the closest points to* e
    G =*scan_convert*(p)
    // Loop over the scan converted grid points.
    for each $(i, j) \in G$*:*
        d_{new} = *distance to* e
        if $|d_{\text{new}}| < |d_{ij}|$*:*
            $d_{ij} = d_{\text{new}}$
            cp_{ij} = *closest point on* e
    end // for
end // for
// Loop over the vertices.
for each $v \in V$*:*
    p =*polygon containing the closest points to* v
    G =*scan_convert*(p)
    // Loop over the scan converted grid points.
    for each $(i, j) \in G$*:*

d_{new} = *distance to* v
if $|d_{\text{new}}| < |d_{ij}|$:
$d_{ij} = d_{\text{new}}$
$cp_{ij} = v$
end // for
end // for
end

Because of the use of floating point arithmetic in representing the polygons, one needs to increase the size of the polygons by a small amount in the outward normal direction. This ensures that grid points which are close to the boundary of the polygon are included and no grid points are left out.

Suppose the curve has M edges and vertices. Let there be N grid points within a distance d of the curve and let r be the ratio of the sum of the areas of all the scan converted polygons divided by the area of the domain within a distance d of the curve. The total computational complexity of the algorithm is $\mathcal{O}(rN+M)$. The $\mathcal{O}(rN)$ term comes from scan conversion and the closest point and distance computations for the grid points. If d is relatively small, (or if d is large and the polygons are effectively clipped), then r will be close to unity. The $\mathcal{O}(M)$ term represents the construction and perhaps the clipping of the polygons.

4.2 Triangle Mesh Surface

We next consider the closest point transform for a triangle mesh surface in 3D. The algorithm is very similar to that for computing the CPT to a piecewise linear curve. Instead of a curve composed of edges and vertices, we will deal with a surface composed of triangular faces, edges and vertices.

For a given grid point, the closest point on the triangle mesh either lies on one of the triangle faces, edges or vertices. Analogous to the polygons containing the grid points which are possibly closest to a given edge or vertex of a curve, in three dimensions we will find polyhedra which contain the grid points which are possibly closest to the faces, edges or vertices. Suppose that the closest point $\boldsymbol{\xi}$ to a grid point $\mathbf{x}$ lies on a triangular face. The vector from $\boldsymbol{\xi}$ to $\mathbf{x}$ is orthogonal to the face. Thus the closest points to a given face must lie within a triangular prism defined by the face and the normal vectors at its three vertices. The prism defined by the face and the outward/inward normals contains the points of positive/negative distance from the face. See Figure 7a for the face polyhedra of an icosahedron.

Consider a grid point $\mathbf{x}$ whose closest point $\boldsymbol{\xi}$ is on a edge. Each edge in the mesh is shared by two faces. The closest points to an edge must lie in a cylindrical wedge defined by the line segment and the normals to the two adjacent faces. If the outside/inside angle between the two adjacent faces is less than π, then there are no points of positive/negative distance from the line segment. See Figure 7b for the edge polyhedra of an icosahedron. Figure 7c shows a single edge polyhedron.

Finally consider a grid point $\mathbf{x}$ whose closest point $\boldsymbol{\xi}$ is on a vertex. Each vertex in the mesh is shared by three or more faces. The closest points to a vertex must lie

in a cone defined by the normals to the adjacent faces. If the mesh is convex/concave at the vertex then there will only be a cone outside/inside the mesh and only points of positive/negative distance. If the mesh is neither convex nor concave at the vertex there are neither positive nor negative cones. Figure 7d shows the vertex polyhedra of an icosahedron.

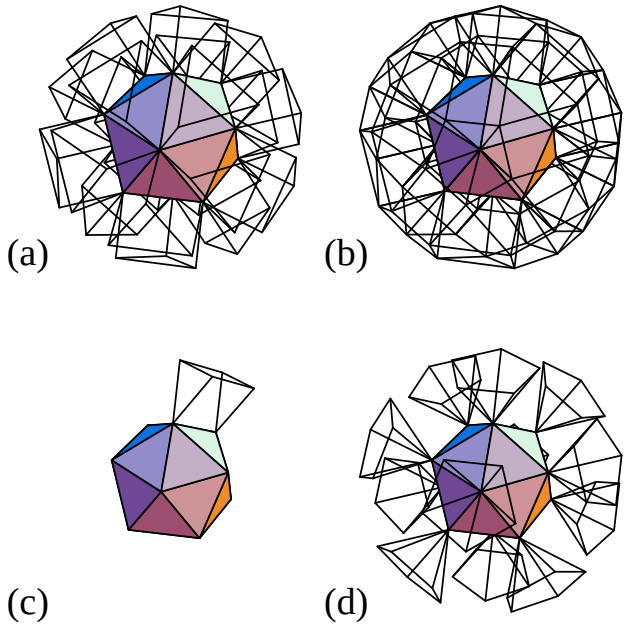


Figure 7: (a) The Positive Polyhedra for the Faces. (b) The Polyhedra for the Edges. (c) The Polyhedron for a Single Edge. (d) The Polyhedra for the Vertices.

We next present a fast algorithm for computing the distance and closest point transform to a triangle mesh surface. Let F be the set of faces, E be the set of edges and V the set of vertices. Let d_{ijk} and cp_{ijk} denote the distance to the surface and the closest point on the surface for the points in a 3D grid.

initially
{ $d_{ijk} = \infty$: *for all* i, j, k }
begin
// Loop over the faces.
for each $f \in F$*:*
    p =*polyhedron containing the closest points to* f

```
        G =scan_convert(p)
        // Loop over the scan converted points.
        for each (i, j, k) ∈ G:
            d_new = distance to e
            if |d_new| < |d_ijk|:
                d_ijk = d_new
                cp_ijk = closest point on f
        end // for
    end // for
    // Loop over the edges.
    for each e ∈ E:
        ...
    end // for
    // Loop over the vertices.
    for each v ∈ V:
        ...
    end // for
    end
```

Let the triangle mesh surface have M faces, edges and vertices. Let the 3D rectilinear grid have N points within a distance d of the surface. Let r be the ratio of the sum of the volumes of all the scan converted polyhedra divided by the volume of the domain within a distance d of the surface. The total computational complexity of the algorithm is $\mathcal{O}(rN + M)$. The $\mathcal{O}(rN)$ again term comes from scan conversion and the closest point and distance computations for the grid points. The $\mathcal{O}(M)$ term represents the construction of the polyhedra.

4.3 The General Algorithm

Consider a manifold is composed of simple shapes, for example, a piecewise cubic curve in 2D or 3D or a surface composed of cubic patches in 3D. One can construct polygons/polyhedra which contain the closest points within a given distance of these shapes. Then one can apply the Characteristics/Scan Conversion algorithm for computing the closest point transform. The algorithm is essentially the same as that presented for the examples of piecewise linear curves and triangle surfaces. The only difference lies in the details of constructing the polygons/polyhedra and computing distance/closest point to the component shapes. For each component, one constructs a polygon/polyhedron which contains all the grid points within a given distance of the component. One then uses scan conversion to determine which grid points are possibly within the given distance of the component. Then the distance to the component and the closest point on the component are computed for these grid points. For more complicated component shapes, such as cubic patches, one would need to solve nonlinear equations to determine the distance and closest point.

4.4 Concurrent Algorithm

We note that the Characteristics/Scan-Conversion algorithm is embarrassingly concurrent. Suppose the grid on which we want to compute the CPT is distributed over a number of processors. Consider computing the closest point transform to a distance d away from a manifold. If each processor has the portion of the manifold that is within a distance d of its grid, then each processor simply executes the sequential algorithm with its portion of the manifold and the grid. Additionally, one can take advantage of multi-threaded concurrency. On any given processor, scan converting and computing closest points for each polygon/polyhedron are independent tasks which can proceed concurrently.

5 Performance of the CPT Algorithm

First we examine the performance of the Characteristics/Scan Conversion algorithm as we vary the grid size. To verify that the algorithm has linear computational complexity in the grid size, we examine execution time as we refine the grid. We compute the closest point transform to a tessellation of the unit sphere with 2048 faces on the domain $(-2,2)\times(-2,2)\times(-2,2)$ to a distance of 0.05 for grid sizes from 10^3 to 200^3. Figure 8 shows a log-log plot of execution time versus grid size along with the line of linear scalability. This shows the linear computational complexity. Initially there is super-linear scalability due to coarser inner loops in the algorithm. As the grid is refined, the polyhedra contain more grid points. Scan converting polyhedra containing many points is more efficient than scan converting polyhedra containing few points.

Next we examine the performance of the Characteristics/Scan Conversion algorithm as we vary the mesh size. To verify that the algorithm has linear computational complexity in the mesh size, we examine execution time as we refine the mesh. We compute the closest point transform on a $100\times100\times100$ grid to tessellations of the unit sphere on the domain $(-1.2,1.2)\times(-1.2,1.2)\times(-1.2,1.2)$ to a distance of 0.1 for mesh sizes from 32 to 131072 faces. Figure 9 shows a log-log plot of execution time versus mesh size along with the line of linear scalability. This shows the linear computational complexity. Initially there is super-linear scalability because the total volume of the polyhedra decreases as the mesh is refined.

5.1 Comparison with Other Methods

5.1.1 Finite Difference Methods

We compare finite difference (FD) methods for computing distance with the CSC algorithm.

- FD methods compute an approximate distance. The CSC algorithm is accurate to within machine precision.

Grid Size	Execution Time (sec)	Grid Size	Execution Time (sec)
10^3	0.55	110^3	3.42
20^3	0.66	120^3	4.04
30^3	0.82	130^3	4.78
40^3	0.97	140^3	5.64
50^3	1.17	150^3	6.56
60^3	1.41	160^3	7.68
70^3	1.70	170^3	8.95
80^3	2.01	180^3	10.11
90^3	2.42	190^3	11.77
100^3	2.86	200^3	13.29

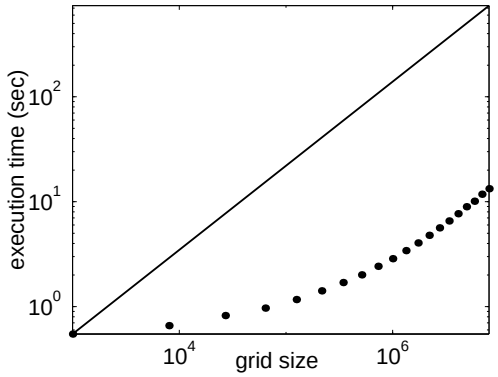


Figure 8: Log-log plot of execution time versus grid size. The line of linear scalability.

# of Faces	Execution Time (sec)
32	0.55
128	0.77
512	1.18
2048	2.42
8192	6.56
32768	21.52
131072	78.05

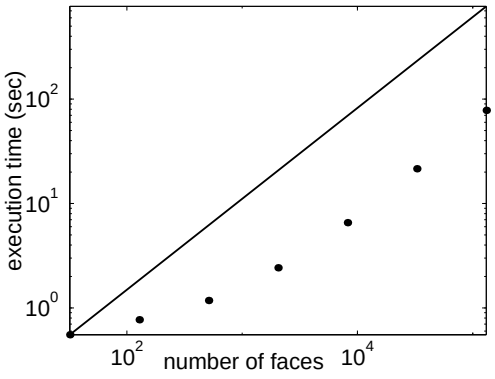


Figure 9: Log-log plot of execution time versus the number of faces. The line of linear speed-up.

- The CSC algorithm computes the closest point. To compute the approximate closest point with a FD method, one first computes the distance and then follows the gradient back to the manifold for each grid point.
- The CSC algorithm takes an explicit representation of the manifold as an initial

condition. FD methods take an implicit representation, i.e. the value of the distance on grid points surrounding the manifold, as an initial condition.

- The computational complexity of the CSC algorithm in linear in both the size of the manifold and the size of the grid, $\mathcal{O}(M+N)$. Once one has generated the initial condition by computing the distance on the grid points close to the manifold, the computational complexity of FD methods is $\mathcal{O}(\alpha N)$ for iterative methods and $\mathcal{O}(N \log N)$ for fast marching methods.

- It is relatively easy to implement FD methods in higher dimensional spaces. An FD method in 10D is little different than one in 2D. It requires significant work to implement the CSC algorithm in higher dimensions because one has to do geometry and scan conversion.

- The CSC algorithm is embarrassingly concurrent; each process runs the sequential algorithm independent of other processes. FD methods require communication between neighboring processes for concurrent implementations.

We compare the performance of the Characteristics/Scan Conversion algorithm and the Fast Marching algorithms used to compute the distance transform to tessellations of a unit sphere to a distance of 0.1 on the domain $(-1.2, 1.2) \times (-1.2, 1.2) \times (-1.2, 1.2)$. The CSC algorithm was run for tessellations with 2048 faces and 32768 faces. (The Fast Marching finite difference method takes an initial condition on the grid so the execution time is independent of the tessellation refinement.) Table 1 gives the execution times in seconds. The CSC algorithm compares favorably with the FM algorithm, especially as the grid is refined. For initial data given explicitly as a surface, (instead of implicitly on the grid), using the CSC algorithm to compute the distance transform would typically be preferable to generating an implicit initial condition and using a finite difference method.

Grid Size	CSC (2048 Faces)	CSC (32768 Faces)	Fast Marching
25^3	0.86	10.39	0.15
50^3	1.26	13.87	2.53
75^3	1.90	17.61	11.35
100^3	2.62	21.42	31.80
125^3	3.67	25.74	71.40
150^3	4.88	30.20	133.56
175^3	6.57	35.19	230.96
200^3	8.42	40.28	376.17
225^3	10.22	46.41	544.37
250^3	12.81	52.40	782.26

Table 1: Comparison of execution times (sec) for computing the distance transform.

5.1.2 LUB-Tree Methods

The LUB-Tree method and the CSC algorithm are both geometrically based and thus have many similarities. They both compute the distance and closest point, accurate to within machine precision. They take an explicit representation of the manifold as an initial condition. Both algorithms are embarrassingly concurrent.

The primary difference between the algorithms is in performance. The computational complexity of the CSC algorithm is $\mathcal{O}(M + N)$, while the LUB-Tree method is $\mathcal{O}(M \log M + N \log M)$. Thus the LUB-Tree method is well suited for computing a small number of closest points but is not efficient in computing the closest point transform. Also, the LUB-Tree method is not easily adapted to computing the distance/closest point for only those grid points within a specified distance of the manifold.

6 Conclusions

We have presented the Characteristics/Scan Conversion algorithm for computing the closest point transform to a manifold. The algorithm utilizes scan conversion to efficiently solve the Eikonal equation with the method of characteristics. The algorithm has optimal computational complexity. In computing the closest point or distance transform to within a given distance of a manifold the CSC algorithm typically performs better than previously developed geometry based methods for computing the closest point transform and finite difference based methods for computing the approximate distance transform.

It may possible to adapt the CSC algorithm to efficiently solve Hamilton-Jacobi equations such as the anisotropic Eikonal equation, $|\nabla u| = f(x, y, z)$. Suppose the initial condition is specified on a manifold that is composed of simple shapes. For each of these shapes, one would compute a number of characteristic curves. Next one would need to determine the volume that contains the characteristic curves emanating from the shape. After scan converting this volume one would use the solution determined on the characteristic curves to extrapolate the solution to the grid points.

7 Acknowledgments

Thanks go to Dan Meiron for advising me on this research and the writing of this paper and to Michael Aivazis for assisting me with the implementation of the algorithms. Support for this work was provided by the Accelerated Strategic Computing Initiative under subcontract number B341492 of DOE contract W-7405-ENG-48.

References

[1] O. Atsuyuki, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Wiley & Sons, New York, 1992.

[2] D.E. Johnson and E. Cohen. A Framework for Efficient Minimum Distance Computations. In *Proc. IEEE Intl. Conf. Robotics & Automation, Leuven, Belgium*, pages 3678–3684, May 1998.

[3] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4), 1987.

[4] E. Morano M. Arienti, P. Hung and J. Shepherd. A Level Set Approach to Eulerian-Lagrangian Coupling. In preparation.

[5] S. Osher and J.A. Sethian. Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[6] J.A. Sethian. A Fast Marching Level Set Method for Monotonically Advancing Fronts. In *Proc. Nat. Acad. Sci.*, volume 94, pages 1591–1595, 4 1996.

[7] Y.R. Tsai. Rapid and Accurate Computation of the Distance Function Using Grids. Technical report, Department of Mathematics, University of California, Los Angeles, 2000.

A Framework for Level Set Segmentation of Volume Datasets

Ross Whitaker
School of Computing
University of Utah

David Breen, Ken Museth, and Neha Soni
Computer Science Department
California Institute of Technology

Abstract

This paper presents a framework for extracting surface models from a broad variety of volumetric datasets. These datasets are produced from standard 3D imaging devices, and are all noisy samplings of complex biological structures with boundaries that have low and often varying contrasts. The level set segmentation method, which is well documented in the literature, creates a new volume from the input data by solving an initial-value partial differential equation (PDE) with user-defined feature-extracting terms. Given the local/global nature of these terms, proper initialization of the level set algorithm is extremely important. Thus, level set deformations alone are not sufficient, they must be combined with powerful initialization techniques in order to produce successful segmentations. Our level set segmentation approach consists of defining a set of suitable pre-processing techniques for initialization and selecting/tuning different feature-extracting terms in the level set algorithm. This collection of techniques forms a toolkit that can be applied, under the guidance of a user, to segment a variety of volumetric data. Users can combine these methods in different ways and thereby access a wide range of functionalities, several of which are described in this paper and demonstrated on noisy volume data.

1 Introduction

As visualization tasks grow in size and complexity, the problem of presenting data effectively is accompanied by another, potentially more difficult problem—how to extract *presentable* data from the flood of raw information produced by large simulations and high resolution instruments. Thus, the field of *data visualization* is intimately tied to more traditional studies of *data analysis* such as signal and image processing, pattern recognition, artificial intelligence, and computer vision. However, in contrast to more conventional areas of data analysis, the field of visualization explicitly *includes the user* in the process of filtering, extracting, and rendering meaningful data.

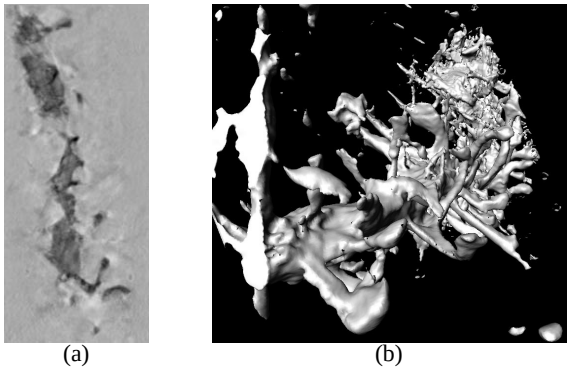
This paper deals with a specific visualization problem—that is, how to build meaningful 3D models of complex structures from noisy datasets generated from 3D imaging devices. In certain circumstances such data can be visualized *directly* [1, 2, 3, 4]. While direct techniques can provide useful insights into volume data, they are insufficient for many problems. For instance, direct volume rendering techniques typically do not remove occluding structures, i.e., they do not allow one to "peel back" the various layers of the data to expose the inner structures that might be of interest. They also do not generate the models needed for quantitative study/analysis of the visualized structures. Furthermore, direct visualization techniques typically do not perform well when applied directly to noisy data, unless one filters the data first. Techniques for filtering noisy data are abundant in the literature, but there is a fundamental limitation—filtering that reduces noise tends to distort the shapes of the objects in the data. The challenge is to find methods which present the best tradeoff between fidelity and noise.

Level set segmentation relies on a surface-fitting strategy, which is effective for dealing with both small-scale noise and smoother intensity fluctuations in volume data. The level set segmentation method, which is well documented in the literature [5, 6, 7, 8], creates a new volume from the input data by solving an initial-value partial differential equation (PDE) with user-defined feature-extracting terms. Given the local/global nature of these terms, proper initialization of the level set algorithm is extremely important. Thus, level set deformations alone are not sufficient, they must be combined with powerful initialization techniques in order to produce successful segmentations. Our level set segmentation approach consists of defining a set of suitable pre-processing techniques for initialization and selecting/tuning different feature-extracting terms in the level set algorithm. We demonstrate that combining several pre-processing steps with level set deformations produces a powerful toolkit that can be applied, under the guidance of a user, to segment a wide variety of volumetric data.

There are more sophisticated strategies for isolating meaningful 3D structures in volume data. Indeed, the so called *segmentation problem* constitutes a significant fraction of the literature in image processing, computer vision, and medical image analysis. For instance, statistical approaches [9, 10, 11, 12] typically attempt to identify tissue types, voxel by voxel, using a collection of measurements at each voxel. Such strategies are best suited to problems where the data is inherently multi-valued or where there is sufficient prior knowledge [13] about the shape or intensity characteristics of the relevant anatomy. Alternatively, anatomical structures can be isolated by grouping voxels based on local image properties. Traditionally, image processing has relied on collections of edges, i.e. high-contrast boundaries, to distinguish regions of different types [14, 15, 16]. Furthermore deformable models, incorporating different degrees of domain-specific knowledge, can be *fitted* to the 3D input data [17, 18]. The work of this paper demonstrates a mathematical and computational framework which effectively combines or unifies classification, filtering, and surface-fitting approaches to modeling and visualizing 3D data.

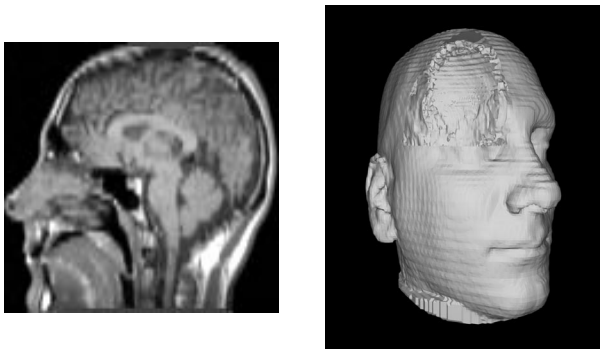
2 Example Datasets

Our work is largely motivated by the desire to produce a semi-automatic segmentation approach which can partly or fully replace the tedious and extremely time-consuming process of manual data segmentation – a solution which to our initial surprise is widely used by colleagues in biology and medicine. Thus, to scientists working in these fields even an approximate scheme which can segment out approximately 90% of the model is immensely useful because it reduces the manual labor needed to produce a final result. We stress that there exists no fully automatic solution to the segmentation problem typically encounter in 3D imaging. For example, Figure 1(a) shows one of 270 slices of an electron tomography (ET) volume of a *spiny dendrite* provided by the National Center for Microscopy and Imaging Research, at UC San Diego. The complex structure of the dendrite and the noisy nature of the data make the rendering of such volume data difficult. Figure 1(b) shows the results of attempting to isolate the relevant structures in this dataset by extracting isosurfaces at greyscale value of 129. For this example we have blurred the data with a small Gaussian ker-



(a) (b)

Figure 1: a) One slice of a $154 \times 586 \times 270$ ET scan of a spiny dendrite shows low contrast and high noise content in a relatively complex data set. b) An isosurface rendering, with prefiltering, shows how noise and inhomogenieties in density interfere with visualizing the 3D structure of the dendrite.



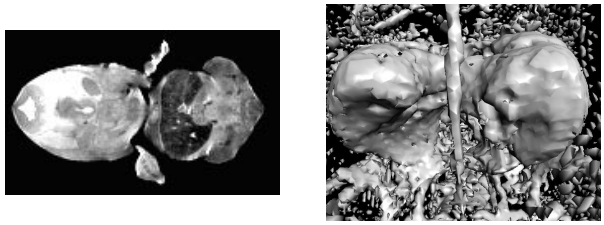
(a) (b)

Figure 2: a) One slice of a $130 \times 128 \times 128$ magnetic resonance (MR) volume of a human head shows high-contrast, relatively noise-free data with numerous internal structures. b) An isosurface rendering, with a small wedge removed for visualization, shows aliasing and internal structures that are not appropriate for the application.

nel ($\sigma = 1.0$) to try to improve the appearance of the isosurfaces. Despite the smoothing the isosurfaces are quite noisy, and contain many small, disconnected pieces that are not indicative of the structure of the dendrite. Furthermore, fluctuations in the tissue density both within and outside of the dendrite create a large number of distortions which prevent the isosurface from accurately representing the underlying shape of that structure.

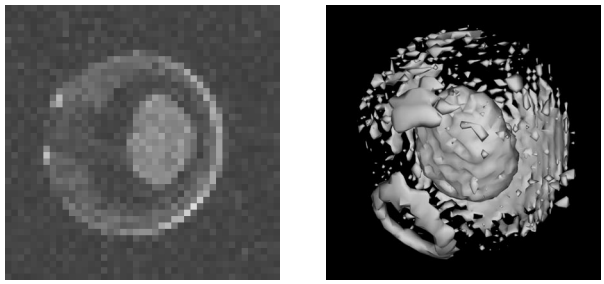
Note that the image shonw in Figure 1(b) is produced in two stages: First, we compute the isosurface with the Marching Cubes algorithm [19] for a given isovalue. Next, the polygonal mesh is displayed using conventional graphics hardware. Alternatively we could visualize structures within the volume data using a one-stage direct method such as volume-rendering (e.g. ray casting with transfer functions or maximum intensity projection). Our choice of Marching Cubes for rendering isosurfaces of this and other datasets in this paper is not essential to the proposed method. The problems of noise and aliasing, present in the examples in this paper, would exist even if we used a direct volume rendering technique.

A second example, shown in Figure 2(a), is a magnetic resonance (MR) scan of a human head. Here the problem is not so much the quality of the data—isosurfaces can be used to visual-



(a) (b)

Figure 3: a) One slice of a $130 \times 128 \times 128$ magnetic resonance (MR) volume of a human head shows high-contrast, relatively noise-free data with numerous internal structures. b) An isosurface rendering, with a small wedge removed for visualization, shows aliasing and internal structures that are not appropriate for the application.



(a) (b)

Figure 4: a) One slice of a $44 \times 45 \times 43$ MR scan of a frog embryo. b) A Marching Cubes isosurface from the frog embryo volume. Isovalue = 60.

(a) (b)

Figure 5: a) Level set models represent curves and surfaces implicitly using greyscale images. For example an ellipse is represented as the level set of an image shown here. b) To change the shape of the ellipse we modify the greyscale values of the image by solving a certain PDE.

ize the skin or skull. In this case the particular application [20] requires a relatively smooth, simple, closed surface, and will not tolerate significant aliasing. The application also requires that the fairly complicated structure of the inner head (usually unseen) be removed. Figure 2(b) shows a isosurface rendering, at a greyscale value of 30, which demonstrates the aliasing in the data. A small wedge has been removed to show the complex internal structures in this volume.

The third example, shown in Figure 3(a), is a $256 \times 128 \times 128$ MR scan of a 12-day-old mouse embryo. Colleagues in the Caltech Biological Imaging Center (BIC) are using such images to develop a detailed atlas for the gestational development of these organisms. For this paper we will consider the specific task of isolating the liver, which is the dark, kidney-shaped area on the right. The liver, however, is not a single grey-scale value, and it is bordered by both more dense and less dense regions. Furthermore, the data contains noise. Therefore, the liver is not easily isolated by simple greyscale classification or isovalue schemes. Figure 3(b) shows an isosurface rendering which accommodates high and low thresholds associated with the liver, i.e., the zero crossings of $I' = \min(I - t_{\text{lo}}, t_{\text{hi}} - I)$, where I is the input volume, and $t_{\text{hi/lo}}$ are the thresholds. The model constructed from the isosurface shows significant artifacts from noise and low-frequency fluctuations in the tissue. It also shows artifacts from the greyscale classification, which captures a large number of voxels in the transition between the skin and the surrounding regions. Smoothing further aggravates this problem.

The final example, shown in Figure 4(a), is a $44 \times 45 \times 43$ *in vivo* MR scan of a frog embryo. This is one slice from one scan of a sequence of 22 volumes taken over a 24-hour period. Colleagues at the Caltech BIC are acquiring time-lapsed MR volume sequences in order to generate the first 3D staging sequence of a developing frog embryo. They require models of the dynamic structures that appear, move, change shape, merge and/or disappear over time within the embryo, as well as the outside shell. The individual structures do not necessarily have distinct signals in the MR scans, thus making it difficult to computationally isolate them. Figure 4(b) presents a polygonal isosurface (isovalue = 60) generated with the Marching Cubes algorithm [19]. At this isovalue two internal structures are produced, as well as a significant part of the outer shell.

3 Level Set Surface Models

When considering deformable models for segmenting 3D volume data, one is faced with a choice from a variety of surface representations, including triangle meshes [20, 21], superquadrics [22, 23, 24], and many others [18, 25, 26, 27, 28, 29, 30]. Another option is an implicit level set model, i.e., specifying the surface as a *level set* of a scalar volumetric function, $\phi : U \mapsto \mathbb{R}$, where $U \subset \mathbb{R}^3$ is the range of the surface model. Thus, a surface S is

$$S = \{s | \phi(s) = k\}, \tag{1}$$

and the choice of the isovalue, k, is arbitrary. In other words, S is the set of points s in $\mathbb{R}^3$ that composes the k isosurface of ϕ. The embedding ϕ can be specified as a regular sampling on a rectilinear grid.

Our overall scheme for segmentation is largely based on the ideas of Osher and Sethian [31] that model propagating surfaces with (time-varying) curvature-dependent speeds. The surfaces are viewed as a specific level set of a higher-dimensional function ϕ – hence the name level set methods. These methods provide the mathematical and numerical mechanisms for computing surface deformations as isovalues of ϕ by solving a partial differential equation on the 3D grid. That is, the level set formulation provides a set of numerical methods that describes how to manipulate the greyscale values in a volume, so that the isosurfaces of ϕ move in a prescribed manner (shown in Figure 5). This paper does not present a comprehensive review of level set methods, but merely introduces the basic concepts and the notation used in successive sections. See [7] for more details.

There are two different approaches to defining deformable surface from a level set of a volumetric function as described in Equation 1. Either one can think of $\phi(s)$ as a *static* function and change the isovalue $k(t)$ or alternatively fix k and let the volumetric function *dynamically* change in time, i.e. $\phi(s,t)$. Thus, we can mathematically express the static and dynamic model respectively as

$$\phi(s) = k(t) \tag{2a}$$
$$\phi(s,t) = k. \tag{2b}$$

To transform these definitions into partial differential equations which can easily be solved by standard numerical techniques, we differentiate both sides of Equation 2 with respect to time t, and apply the chain rule:

$$\nabla\phi(s)\frac{\mathrm{d}s}{\mathrm{d}t} = \frac{\mathrm{d}k(t)}{\mathrm{d}t} \tag{3a}$$
$$\frac{\partial\phi(s,t)}{\partial t} + \nabla\phi(s,t) \cdot \frac{\mathrm{d}s}{\mathrm{d}t} = 0 \tag{3b}$$

The static Equation 3a is often referred to as the "Eikonal" equation and defines a boundary value problem for the time-independent volumetric function ϕ. This static level set approach has been solved [32, 33] using a "Fast Marching Method". However, it has some inherent limitations following the simple definition in Equation 2a. Because ϕ is a function (i.e. single-valued), isosurfaces cannot self intersect over time, i.e. shapes defined in the static model are strictly expanding or contracting over time. The dynamic level set approach of Equation 3b is much more flexible and shall serve as the basis of the segmentation scheme in this paper. Equation 3b is sometimes referred to as a "Hamilton-Jacobi-type" equation and defines an initial-value problem for the time-dependent ϕ. Throughout the remainder of this paper we shall for simplicity refer to this dynamical approach as the level set method – and completely ignore the static alternative.

Thus, to summarize the essence of the (dynamic) level set approach; let $\mathrm{d}s/\mathrm{d}t$ be the movement of a point on a surface as it deforms, such that it can be expressed in terms of the position of $s \in U$ and the geometry of the surface at that point, which is, in turn, a differential expression of the implicit function, ϕ. This gives a partial differential equation on ϕ: $s \equiv s(t)$

$$\frac{\partial\phi}{\partial t} = -\nabla\phi \cdot \frac{\mathrm{d}s}{\mathrm{d}t} \equiv -\nabla\phi \cdot F(s, \mathrm{D}\phi, \mathrm{D}^2\phi, \ldots), \tag{4}$$

where F is a user-defined "speed" term which depends on a set of order-n derivatives of ϕ, $D^n\phi$, evaluated at s, as well as other functions of s. Because this relationship applies to every level set of ϕ, i.e. all values of k, this equation can be applied to all of U, and therefore the movements of *all* the level set surfaces embedded in ϕ can be calculated from Equation 4. Such level set methods are well documented in the literature for applications such as computational physics [34], image processing [35, 36], computer vision [6, 37], medical image analysis [6, 38], shape morphing[39], and 3D reconstruction [40].

The level set representation has a number of practical and theoretical advantages over conventional surface models, especially in the context of deformation and segmentation. First, level set models are topologically flexible, they easily represent complicated surface shapes that can, form holes, split to form multiple objects, or merge with other objects to form a single structure. These models can incorporate many (millions) of degrees of freedom, and therefore they can accommodate complex shapes such as the dendrite in Figure 1.

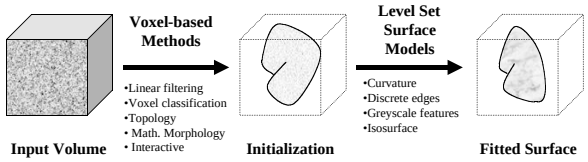


Figure 6: Level set segmentation stages – initialization and surface deformation.

Indeed, the shapes formed by the level sets of ϕ are restricted only by the resolution of the sampling. Thus, there is no need to reparameterize the model as it undergoes significant changes in shape.

Level set methods have been shown to be effective in extracting surface structures from biological and medical data. For instance Malladi et al. [6] propose a method in which the level sets form an expanding or contracting contour which tends to "cling" to interesting features in 2D angiograms. At the same time the contour is also influenced by its own curvature, and therefore remains smooth. Whitaker has shown [38, 41, 42] that level sets can be used to simulate conventional deformable surface models, and demonstrated this by extracting skin and tumors from *thick-sliced* (e.g. clinical) MR data, and by reconstructing a fetal face from 3D ultrasound. A variety of authors [36, 43, 44] have presented variations on the method with results for 2D and 3D data. Sethian [7] gives several examples of level set curves and surface for segmenting CT and MR data.

The purpose of this paper is to present a collection of initialization and level set mechanisms which form a "toolbox" for volume dataset segmentation. We also show how these methods can be combined to solve the problems presented in Figures 1–4. These tools provide a set of techniques that are not as direct as simple thresholding or volume rendering but are more powerful than the "hand-contouring" that is currently the state-of-the-art in many applications, such as the dendrite example in Figure 1.

4 Segmentation Stages

Our level set segmentation process has two major stages, initialization and level set surface deformation, as seen in Figure 6. Each stage is equally important for generating a correct segmentation. Within our framework a variety of operations are available in each stage. A user must "mix-and-match" these operations in order to produce the desired result.

4.1 Initialization

Because the deformable models move using gradient descent, they seek *local solutions*, and therefore the results are strongly dependent on the initialization, i.e., the starting position of the surface. Thus, one controls the nature of the solution by specifying an initial model from which the surface deformation process proceeds. We have implemented both computational (i.e. "semi-automated") and manual/interactive initialization schemes; each offers distinct advantages in different situations.

4.1.1 Computational Initialization

Because the level set modeling technology is based on the isosurfaces of volumes, we can, for many different types of problems, computationally construct reasonable initial estimates directly from the input data. We do this by combining a variety of techniques.

Linear filtering: We can filter the input data with a low-pass filter (e.g. Gaussian kernel) to blur the data and thereby reduce noise. This tends to distort shapes, but the initialization need only be approximate.

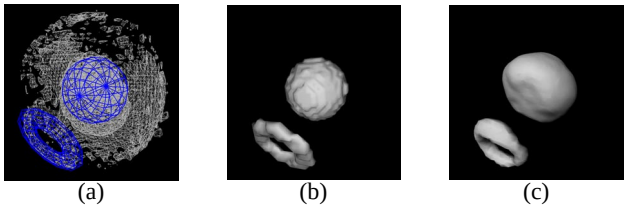
Voxel classification: We can classify pixels based on the filtered values of the input data. For greyscale images, such as those used in this paper, the classification is equivalent to high and low thresholding operations. These operations are usually accurate to only voxel resolution (see [12] for alternatives), but the deformation process will achieve sub-voxel results.

Topological/logical operations: This is the set of basic voxel operations that takes into account position and connectivity. It includes unions or intersections of voxel sets to create better initializations. These logical operations can also incorporate user-defined primitives. Topological operations consist of connected-component analyses (e.g. flood fill) to remove small pieces or holes from objects.

Morphological filtering: This includes binary and greyscale morphological operators on the initial voxel set. For the results in the paper we implement openings and closings using *morphological propagators* [45, 46] implemented with level set surface models. This involves defining offset surfaces of ϕ by expanding/contracting a surface according to the following PDE,

$$\frac{\partial \phi}{\partial t} = \pm |\nabla \phi|, \tag{5}$$

up to a certain time t. The value of t controls the offset distance from the original surface of $\phi(t=0)$. A dilation of size α, D_α, corresponds to the solution of Equation 5 at $t=\alpha$ using the positive sign, and likewise erosion, E_α, uses the negative sign. One can now define a morphological opening operator O_α by first applying an erosion followed by a dilation of ϕ, i.e. $\mathrm{O}_\alpha\phi = \mathrm{D}_\alpha \circ \mathrm{E}_\alpha\phi$, which removes small pieces or thin appendages. A closing is defined as $\mathrm{C}_\alpha\phi = \mathrm{E}_\alpha \circ \mathrm{D}_\alpha\phi$, and closes small gaps or holes within objects. Both operations have the qualitative effect of low-pass filtering the isosurfaces in ϕ—an opening by removing material and a closing by adding material. Both operations tend to distort the shapes of the surfaces on which they operate, which is acceptable for the initialization because it will be followed by a surface deformation.



(a) (b) (c)

Figure 7: a) *[color]* Interactively positioning a CSG model relative to a Marching Cubes mesh. b) Isosurface of a binary scan conversion of the initialization CSG model. c) Final internal embryo structures.

4.1.2 Interactive Initialization

Computational initialization may not always produce a reasonable starting model that deforms into an acceptable final result. Such is the case with the frog-embryo data shown in Figure 4. For volumes that do not allow one to automatically generate an initial model, it

is desirable and easier for the user to interactively specify the initial model which is then deformed to fit to the input data. The interactive initialization process has four steps and is presented in Figures 4(b), and 7(a-c). First, the user generates a Marching Cubes mesh from the input volume. This gives some indication of the structures present in the data (Figure 4(b)). The user then creates a Constructive Solid Geometry (CSG) model which defines the shape of the initial surface. The CSG model in blue is interactively positioned relative to the Marching Cubes mesh (Figure 7(a)). The CSG model is scan-converted into a binary volume, with voxels simply marked as inside (1) or outside (0), using standard CSG evaluation techniques [47] within our modeling system [48]. An isosurface of the initialization volume dataset generated from the torus and sphere in Figure 7(a) is presented in Figure 7(b), isovalue = 0.5. This volume dataset is then used as the starting model for the level set deformation stage, which produces the final result seen in Figure 7(c).

4.2 Level Set Surface Deformation

The initialization should position the model near the desired solution while retaining certain properties such as smoothness, connectivity, etc. Given a rough initial estimate, the surface deformation process moves the surface model toward specific features in the data. One must choose those properties of the input data to which the model will be attracted and what role the shape of the model will have in the deformation process. Typically, the deformation process combines a data term with a smoothing term, which prevents the solution from fitting too closely to noise-corrupted data. There are a variety of surface-motion terms that can be used in succession or simultaneously, in a linear combination to form $F(x)$ in Equation 4.

Curvature: This is the smoothing term. For the work presented here we use the mean curvature of the isosurface H to form a vector in the direction of the surface normal n given by

$$F(x) = Hn = \left(\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}\right)\frac{\nabla\phi}{|\nabla\phi|}. \tag{6}$$

The mean curvature is also the normal variation of the surface area (i.e., minimal surface area). There are a variety of options for second-order smoothing terms [41], and the question of efficient, effective higher-order smoothing terms is the subject of on-going research [7]. For the work in this paper, we combine mean curvature with one of the following three terms, weighting it by a factor β, which is tuned to each specific application.

Edges: Conventional edge detectors from the image processing literature produce sets of "edge" voxels that are associated with areas of high contrast. For this work we use a gradient magnitude threshold combined with non-maximal suppression, which is a 3D generalization of the method of Canny [16]. The edge operator typically requires a scale parameter and a gradient threshold. For the scale, we use small, Gaussian kernels with standard deviation $\sigma = [0.5 - -1.0]$ voxel units. The threshold depends on the contrast of the volume. The distance transform on this edge map produces a volume that has minima at those edges. The gradient of this volume produces a field that attracts the model to these edges. The edges are limited to voxel resolution because of the mechanism by which they are detected. Although this fitting is not sub-voxel accurate, it has the advantage that it can pull models toward edges from significant distances, and thus inaccurate initial estimates can be brought into close alignment with high-contrast regions, i.e. edges, in the input data. If $\mathscr{E}$ is the set of edges, and $D_{\mathscr{E}}(x)$ is the distance transform to those edges, then the movement of the surface model is given by

$$F(x) = \nabla D_{\mathscr{E}}(x). \tag{7}$$

Greyscale features—gradient magnitude: Surface models can also be attracted to certain greyscale features in the input data. For instance, the gradient magnitude indicates areas of high contrast in volumes. By following the gradient of such greyscale features, surface models are drawn to minimum or maximum values of that feature. Typically greyscale features, such as the gradient magnitude are computed with a scale operator, e.g., a derivative-of-Gaussian kernel. If models are properly initialized, they can move according to the *gradient of the gradient magnitude* and settle onto the edges of an object at a resolution that is finer than the original volume.

If $G(x)$ is some greyscale feature, for instance $G(x) = |\nabla I(x)|$, where $I(x)$ is the input data (appropriately filtered—we use Gaussian kernels with $\sigma \approx 0.5$), then

$$F(x) = \pm\nabla G(x), \tag{8}$$

where a positive sign moves surfaces towards maxima and the negative sign towards minima.

Isosurface: Surface models can also expand or contract to conform to isosurfaces in the input data. To a first order approximation, the distance from a point $x \in U$ to the k-level surface of I is given by $(I(x)-k)/|\nabla I|$. If we let $g(\alpha)$ be a fuzzy threshold, e.g., $g(\alpha) = \alpha/\sqrt{1+\alpha^2}$, then

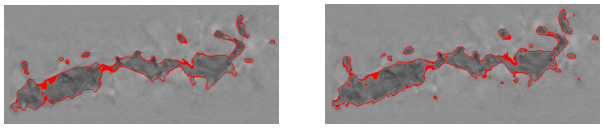
$$F(x) = \frac{\nabla\phi}{|\nabla\phi|} g\left(\frac{I(x)-k}{|\nabla I|}\right) \tag{9}$$

causes the surfaces of ϕ to expand or contract to match the k isosurface of I. This term combined with curvature or one of the other fitting terms can create "quasi-isosurfaces" that also include other considerations, such as smoothness or edge strength.

5 Results

This section describes how our approach may be used to extract structures from the data described in Section 2. We present surface renderings of the resulting models and detail the specific methods needed to construct each model.

Figure 10 shows 3D renderings of the sequence of steps performed on the ET dendrite data from Figure 1. The first two are the initialization steps, generating a smoothed isosurface and filling gaps with topological and morphological operations. The second two are surface deformation steps, first fitting to discrete edges and then to the gradient magnitude. Figure 8 shows a slice with the boundary of the solution drawn in red, that confirms the accuracy of the results—the red boundary is only an indicator of the solution because it is limited to voxel resolution while the level set model has sub-voxel resolution. This figure also shows the same result for a smoothed isosurface—which is significantly affected by density fluctuations in the data. Figures 9 and 11 show the results of the proposed method compared to the results of a manual segmentation, which took approximately 10 hours of slice-by-slice hand contouring. The manual method suffers from slice-wise artifacts, and, because of the size and complexity of the dataset, the manual segmentation is unable to capture the level of detail that we obtain with the surface-fitting results. Manual segmentation can, however, form connections that are not well supported by the data in order to complete the "spines" that cover this dendrite. These types of



(a) (b)

Figure 8: *[color]* a) Voxel-resolution contours of a dendrite using our level set approach. b) Voxel-resolution contours of the isosurface of the smoothed sampling of the same dendrite.

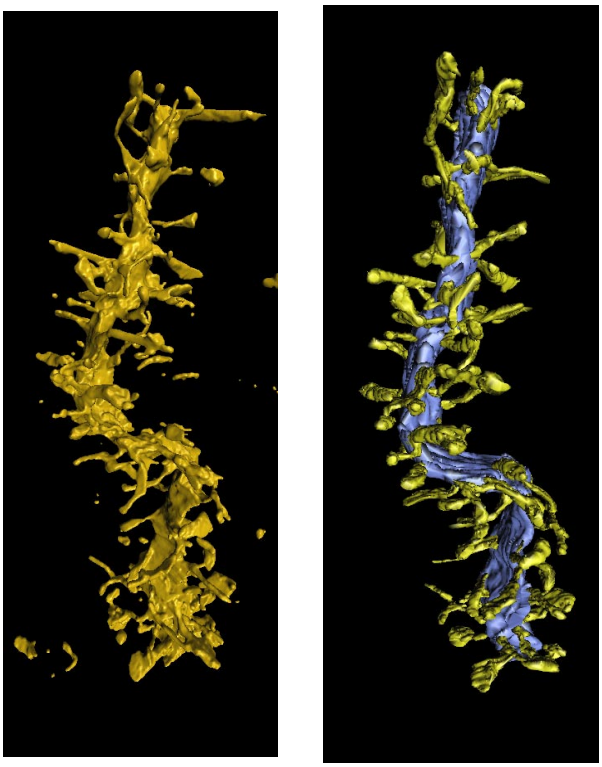
"judgments" that humans make when they perform such tasks by hand are a mixed blessing. Humans can use high-level knowledge about the problem to fill in where the data is weak, but the expectations of a trained operator can interfere with seeing unexpected or unusual features in the data. Our future work will attempt to incorporate user input to guide the surface-fitting results to obtain a better blend of user expectations and data-driven modeling.

Figure 12 shows the results of fitting a surface model to the MR head data shown in Figure 2. Figure 12(a) is a rendering of the initial model which is the result of smoothing the data, using a flood fill on the exterior to remove isolated holes or bubbles within the head, and treating the model with a closing, $C_{5.0}$. Figure 12(b) shows the results of fitting to the isosurface with a curvature term to ensure smoothness. Some detail is lost around the lips and ears, but overall the fidelity is good and the smoother, simpler surface model suites our application quite well [20].

Figure 13 presents 3D renderings of the sequence of steps performed on the mouse MR data from Figure 3. The first step is the initialization, and the second two are the surface deformation, first fitting to discrete edges and then to the gradient magnitude. This is a significant improvement over the result in Figure 3(b) which suffers from noise and misclassifications. Figure 13(d) presents several other structures that were segmented from the mouse embryo dataset. The skin (grey) and the liver (blue) were isolated using computational initialization. The brain ventricles (red) and the eyes (green) were segmented with interactive initialization.

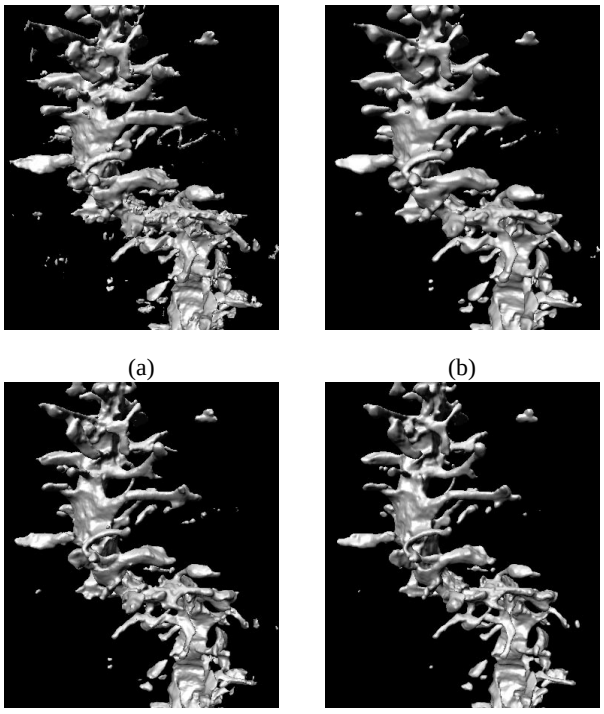
Figure 14 presents models from four samples of the MR series of the developing frog embryo. The top left image (Hour 9) shows the first evident structure, the blastocoel, in blue, surrounded by the outside casing of the embryo in grey. The top right image (Hour 16) demonstrates the expansion of the blastocoel and the development of the blastoporal lip in red. In the bottom left image (Hour 20) the blastoporal lip has collapsed, the blastocoel has contracted, and the archenteron in green has developed. In the bottom right image (Hour 30) the blastocoel has collapsed and only the archenteron is present. As can been seen from Figure 4(b) that it may be difficult to isolate structures using only their voxel values. We therefore used our interactive techniques to isolate (during initialization) most of the structures in the frog embryo samples.

Table 1 describes for each dataset the specific techniques and parameters we used for the results in this paper. These parameters were obtained by first making a sensible guess based on the contrasts and sizes of features in the data and then using trial and error to obtain acceptable results. Each dataset was processed between 4 and 8 times to achieve these results. More tuning could improve things further, and once these parameters are set, they work moderately well for similar modalities with similar subjects. The method is iterative, but the update times are proportional to the surface area. On an SGI 180MHz MIPS 10000 machine, the smaller mouse MR dataset required approximately 10 minutes of CPU time, and the dendrite dataset ran for approximately 45 minutes. Most of this time was spent in the initialization (which requires several complete passes through the data) and in the edge detection. The frog



(a) (b)

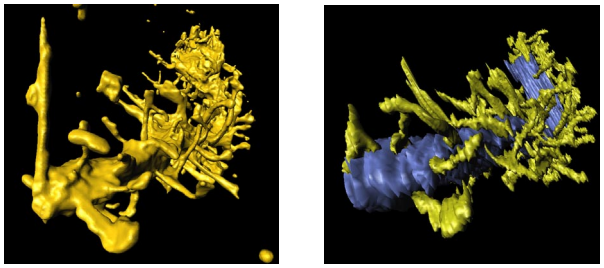
Figure 9: *[color]* a) Rendering of a dendrite segmented using our the proposed method. b) Rendering of the same dendrite, but this time segmented manually.



(a) (b)

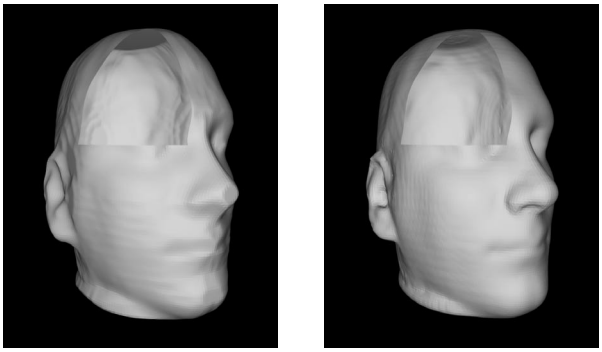
(c) (d)

Figure 10: a) The steps in the surface fitting process: An isosurface of smoothed data. b) Morphological operators fill in gaps and remove smaller, disconnected pieces. c) Fitting to edges brings the model closer to high-contrast regions in the data. d) Fitting to maximal gradient magnitude gives more detail.



(a) (b)

Figure 11: *[color]* a) Close-up view of the final result of the dendrite rendering using our scheme – note the level of details. b) Close-up view of the manual segmentation – note the lack of detail compared to the proposed method.



(a) (b)

Figure 12: a) This image shows the rough initialization surface used for a level set segmentation of an MR scanned head. A small section of the surface has been removed to show that it does not contain internal structures. b) The final result is smoothed (almost no aliasing from the scanning slices), but with good fidelity.

embryo datasets needed only a few minutes of processing time, because they did not require computational initialization and are significantly smaller than the other example datatsets.

6 Conclusions

This paper describes a system that uses level set surface models in conjunction with a suite of initialization techniques to segment structures in volume data. Level set surface modeling is a technology that allows one to manipulate or deform the isosurfaces of a volume toward interesting features in the input data. Because the technology is volumetric, it provides opportunities to combine voxel-based techniques, such as filtering, classification, and morphology with surface-fitting methods based on deformable models. We have shown that combining level set methods with a variety of initialization techniques produces a powerful framework capable of segmenting many different types of volume datasets. In the case of the ET dendrite data, our approach offers significant advantages in both time and quality over hand-contoured segmentations, which are currently the state-of-the-art.

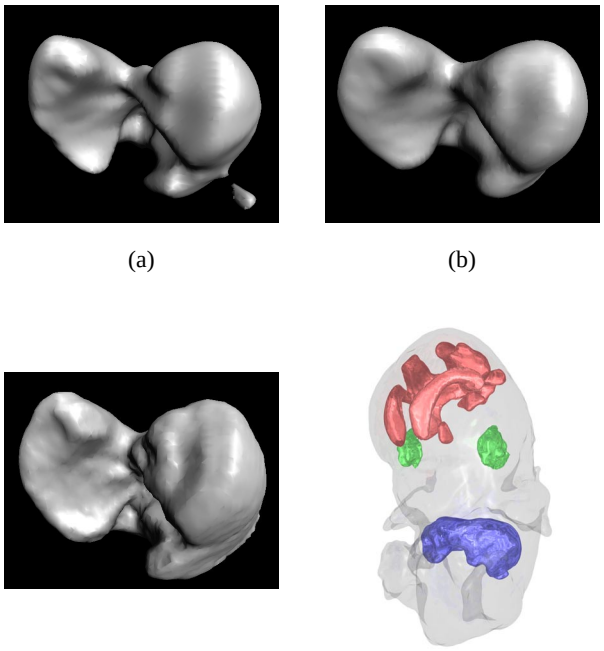
Currently there are two significant drawbacks of the proposed method. First is the choice of parameters. There are a number of parameters that must be tuned, and their settings affect the final solution. The second drawback is the computation time, which is quite long for large datasets. The second problem aggravates the first, because exploring the parameter space by trial and error is a potentially lengthy process. Future work will focus on increasing the update rates by parallelizing the computation. This is feasible because the numerical methods lend themselves to a spatial decomposition of the model domain. If the updates were sufficiently fast, users can explore the parameter space interactively by turning various knobs and evaluating the quality of the results. This would greatly increase the effectiveness of the method.

Acknowledgements

We would like to thank Dr. Alan Barr for his support of this project, and Dr. Cyrus Papan, Dr. Russ Jacobs, Dr. Eric Ahrens, Dr. Mark Ellisman, Dr. Maryanne Martone and Mr. David Weinstein for their assistance and cooperation. This work was supported

Dataset	Initialization	Surface Fitting
Dendrite	1. Gaussian blur $\sigma = 0.5$ 2. Threshold: $I < 127$ 3. Fill isolated holes 4. Morphology: $O_{0.5} \circ C_{1.5}$	1. Edge fitting: $\sigma = 0.75$, threshold = 6, $\beta = 0.1$ 2. Gradient magnitude fitting: $\sigma = 0.5$, $\beta = 1.0$
Head	1. Gaussian blur $\sigma = 1.0$ 2. Threshold: $I > 30$ 3. Fill isolated holes 4. Morphology: $C_{5.0}$	1. Isosurface fitting: $\beta = 5.0$, $k = 30$.
Mouse	1. Gaussian blur $\sigma = 0.5$ 2. Threshold: $I > 3, I < 60$ 3. Fill isolated holes 4. Morphology: $O_{2.0} \circ C_{3.0}$	1. Edge fitting: $\sigma = 0.75$, threshold = 20, $\beta = 2$ 2. Gradient magnitude fitting: $\sigma = 0.5$, $\beta = 16.0$
Frog	1. Interactive	1. Gradient magnitude fitting: $\sigma = 1.25$, $\beta = 1.0$

Table 1: Parameters for processing example datasets.



(a) (b) (c) (d)

Figure 13: a) The initialization of a mouse liver dataset using morpholog to remove small pieces and holes. b) Surface fitting to discrete edges. c) The final fit to maxima of gradient magnitude. d) *[color]* Final mouse embryo model with skin (grey), liver (blue), brain ventricles (red), and eyes (green).

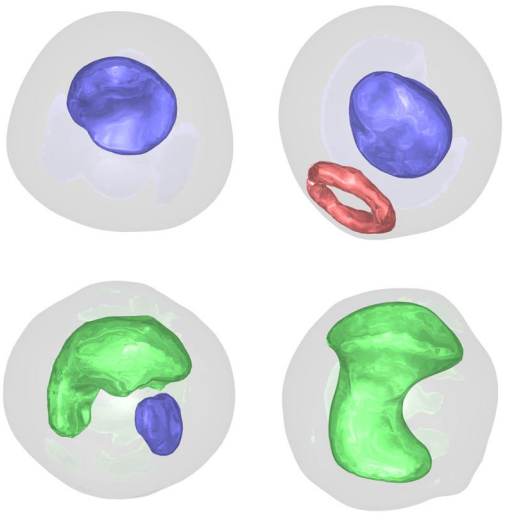


Figure 14: *[color]* Geometric structures extracted from MRI scans of a developing frog embryo, with blastocoel (blue), blastoporal lip (red), and archenteron (green). Hour 9 (top left). Hour 16 (top right). Hour 20 (bottom left). Hour 30 (bottom right).

by National Science Foundation grants #ASC-89-20219, #ACI-99-82273, and #ACI-00-89915, Office of Naval Research grant #N00014-01-10033, the National Institute on Drug Abuse, the National Institute of Mental Health and the NSF, as part of the Human Brain Project, the National Library of Medicine "Insight" Project #N01-LM-0-3503, and the Caltech SURF Program. The datasets are courtesy of the National Center for Microscopy and Imaging Research (funded by NIH grant #P41-RR04050), at UC San Diego (Figure 1), the University of Utah SCI Institute (Figure 2), and the Caltech Biological Imaging Center (Figures 3 and 4).

References

[1] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *SIGGRAPH '88 Proceedings*, pp. 65–74, August 1988.

[2] M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, vol. 9, no. 3, pp. 245–261, 1990.

[3] D. Laur and P. Hanrahan, "Hierarchical splatting: A progressive refinement algorithm for volume rendering," in *SIGGRAPH '91 Proceedings* (T. W. Sederberg, ed.), pp. 285–288, July 1991.

[4] S. Parker, M. Parker, Y. Livnat, P. Sloan, C. Hansen, and P. Shirley, "Interactive ray tracing for volume visualization," *IEEE Trans. on Viz. and Comp. Graph.*, vol. 5, no. 3, pp. 238–250, 1999.

[5] M. Leventon, O. Faugeraus, W. Grimson, and W. Wells III, "Level set based segmentation with intensity and curvature priors," in *Workshop on Mathematical Methods in Biomedical Image Analysis Proceedings*, pp. 4–11, June 2000.

[6] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 2, pp. 158–175, 1995.

[7] J. Sethian, *Level Set Methods and Fast Marching Methods*. Cambridge: Cambridge University Press, second ed., 1999.

[8] L. Staib, X. Zeng, R. Schultz, and J. Duncan, "Shape constraints in deformable models," in *Handbook of Medical Imaging* (I. Bankman, ed.), ch. 9, pp. 147–157, Academic Press, 2000.

[9] Z. Wu, H.-W. Chung, and F. W. Wehrli, "A Bayesian approach to subvoxel tissue classification in NMR microscopic images of trabecular bone," *Journal of Computer Assisted Tomography*, vol. 12, no. 1, pp. 1–9, 1988.

[10] Y.-H. Kao, J. A. Sorenson, and S. S. Winkler, "MR image segmentation using vector decompsition and probability techniques: A general model and its application to dual-echo images," *Magnetic Resonance in Medicine*, vol. 35, pp. 114–125, 1996.

[11] H. E. Cline, W. E. Lorensen, R. Kikinis, and F. Jolesz, "Three-dimensional segmentation of MR images of the head using probability and connectivity," *Journal of Computer Assisted Tomography*, vol. 14, pp. 1037–1045, Nov., Dec. 1990.

[12] D. H. Laidlaw, K. W. Fleischer, and A. H. Barr, "Partial-volume Bayesian classification of material mixtures in MR volume data using voxel histograms," *IEEE Transactions on Medical Imaging*, vol. 17, pp. 74–86, feb 1998.

[13] V. E. Johnson, "A framework for incorporating structural prior information into the estimation of medical images," in *Information Processing in Medical Imaging (IPMI'93)* (H. H. Barrett and A. F. Gmitro, eds.), no. 687 in Lecture Notes in Computer Science, pp. 307–321, Springer-Verlag, 1993.

[14] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London*, vol. B, no. 207, pp. 187–217, 1980.

[15] D. Marr, *Vision*. San Francisco: Freeman, 1982.

[16] J. Canny, "A computational approach to edge detection," *IEEE Trans. on Pat. Anal. and Mach. Intel.*, vol. 8, no. 6, pp. 679–698, 1986.

[17] T. Cootes, A. Hill, C. Taylor, and J. Haslam, "The use of active shape models for locating structures in medical images," in *Information Processing in Medical Imaging (IPMI'93)* (H. H. Barrett and A. F. Gmitro, eds.), no. 687 in Lecture Notes in Computer Science, pp. 33–47, Springer-Verlag, 1993.

[18] G. Stetten and S. Pizer, "Medial node models to identify and measure objects in real-time 3d echocardiography," *IEEE Transactions on Medical Imaging*, vol. 18, no. 10, pp. 1025–1034, 1999.

[19] W. Lorenson and H. Cline, "Marching Cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1982.

[20] Z. Wood, M.Desbrun, P. Schröder, and D. Breen, "Semi-regular mesh extraction from volumes," in *Proceedings of Visualization 2000*, pp. 275–282, 2000.

[21] J. Miller, D. Breen, W. Lorensen, R. O'Bara, and M. Wozny, "Geometrically deformed models: A method for extracting closed geometric models from volume data," in *SIGGRAPH '91 Proceedings*, pp. 217–226, July 1991.

[22] A. P. Pentland, "Perceptual organization and the representation of natural form," *Artificial Intelligence*, vol. 28, pp. 293–331, 1986.

[23] D. Terzopoulos and D. Metaxas, "Dynamic 3D models with local and global deformations: Deformable superquadrics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 7, pp. 703–714, 1991.

[24] A. Gupta and R. Bajcsy, "Volumetric segmentation of range images of 3D objects using superquadric models," *CVGIP: Image Understanding*, vol. 58, no. 3, pp. 302–326, 1993.

[25] S. Muraki, "Volumetric shape description of range data using "blobby model"," in *SIGGRAPH '91 Proceedings* (T. W. Sederberg, ed.), pp. 227–235, July 1991.

[26] R. Szeliski, D. Tonnesen, and D. Terzopoulos, "Modeling surfaces of arbitrary topology with dynamic particles," in *Proc. Fourth Int. Conf. on Comp. Vision (ICCV'93)*, (Berlin, Germany), pp. 82–87, IEEE Computer Society Press, May 1993.

[27] T. McInerney and D. Terzopoulos, "A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4d image analysis," *Computerized Medical Imaging and Graphics*, vol. 19, no. 1, pp. 69–83, 1995.

[28] J. Park, D. Metaxas, A. A. Young, and L. Axel, "Deformable models with parameter functions for cardiac motion analysis from tagged MRI data," *IEEE Transactions on Medical Imaging*, vol. 15, pp. 278–289, June 1996.

[29] D. DeCarlo and D. Metaxas, "Shape evolution with structural and topological changes using blending," *IEEE Trans. on Pat. Anal. and Mach. Intel.*, vol. 20, pp. 1186–1205, November 1998.

[30] R. Ramamoorthi and J. Arvo, "Creating generative models from range images," in *SIGGRAPH '99 Proceedings*, pp. 195–204, August 1999.

[31] S. Osher and J. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, pp. 12–49, 1988.

[32] J. Sethian, "A fast marching level set method for monotonically advancing fronts," in *Proceedings of the National Academy of Science*, vol. 93 of 4, pp. 1591–1595, 1996.

[33] J. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1528–1538, 1995.

[34] R. Fedkiw, R. Aslam, R. Merriman, and S. Osher, "A non-oscillatory eulerian approach to interfaces in multimaterial flows (the ghost fluid method)," *Journal of Computational Physics*, vol. 152, pp. 457–492, 1999.

[35] L. Alvarez and J.-M. Morel, "A morphological approach to multiscale analysis: From principles to equations," in *Geometry-Driven Diffusion in Computer Vision* (B. M. ter Haar Romeny, ed.), pp. 4–21, Kluwer Academic Publishers, 1994.

[36] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," in *Fifth Int. Conf. on Comp. Vision*, pp. 694–699, IEEE, IEEE Computer Society Press, 1995.

[37] B. B. Kimia and S. W. Zucker, "Exploring the shape manifold: the role of conservation laws.," in *Shape in Picture: the mathematical description of shape in greylevel images* (Y.-L. O, A. Toet, H. Heijmans, D. H. Foster, and P. Meer, eds.), Springer-Verlag, 1992.

[38] R. T. Whitaker and D. T. Chen, "Embedded active surfaces for volume visualization," in *SPIE Medical Imaging 1994*, (Newport Beach, California), 1994.

[39] R. Whitaker and D. Breen, "Level-set models for the deformation of solid objects," in *The Third International Workshop on Implicit Surfaces*, pp. 19–35, Eurographics, 1998.

[40] R. T. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int. Jrnl. of Comp. Vision*, vol. October, no. 3, pp. 203–231, 1998.

[41] R. T. Whitaker, "Volumetric deformable models: Active blobs," in *Visualization In Biomedical Computing 1994* (R. A. Robb, ed.), (Mayo Clinic, Rochester, Minnesota), pp. 122–134, SPIE, 1994.

[42] R. T. Whitaker, "Algorithms for implicit deformable models," in *Fifth Intern. Conf. on Comp. Vision*, IEEE, IEEE Computer Society Press, 1995.

[43] A. Yezzi, S. Kichenassamy, A. Kumar, P. Olver, and A. Tannenbaum, "A geometric snake model for segmentation of medical imagery," *IEEE Transactions on Medical Imaging*, vol. 16, pp. 199–209, April 1997.

[44] L. Lorigo, O. Faugeraus, W. Grimson, R. Keriven, and R. Kikinis, "Segmentation of bone in clinical knee MRI using texture-based geodesic active contours," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI '98)* (W. Wells, A. Colchester, and S. Delp, eds.), pp. 1195–1204, October 1998.

[45] R. van den Boomgaard and A. W. M. Smeulders, "The morphological structure of images, the differential equations of morphological scale-space," *IEEE Trans. on Pat. Anal. and Mach. Intel.*, vol. 16, no. 11, pp. 1101–1113, 1994.

[46] P. Maragos, "Differential morphology and image processing," *IEEE Transactions on Image Processing*, vol. 5, pp. 922–937, June 1996.

[47] A. Requicha and H. Voelcker, "Boolean operations in solid modeling: Boundary evaluation and merging algorithms," *Proceedings of the IEEE*, vol. 73, no. 1, pp. 30–44, 1985.

[48] P. Getto and D. Breen, "An object-oriented architecture for a computer animation system," *The Visual Computer*, vol. 6, pp. 79–92, March 1990.

A Level-Set Approach for the Metamorphosis of Solid Models

David E. Breen† Ross T. Whitaker‡

†Computer Graphics Laboratory
MS 348-74
California Institute of Technology
Pasadena, CA 91125
Tel (626) 395-2866
FAX (626) 793-9544
david@gg.caltech.edu
corresponding author

‡School of Computing
4540 Merrill Engineering Building
University of Utah
Salt Lake City, UT 84112-9205
Tel (801) 587-9549
FAX (801) 581-5843
whitaker@cs.utah.edu

October 30, 2000

Abstract

This paper presents a new approach to 3-D shape metamorphosis. We express the *interpolation* of two shapes as a process where one shape deforms to maximize its similarity with another shape. The process incrementally optimizes an objective function while deforming an implicit surface model. We represent the deformable surface as a level set (iso-surface) of a densely sampled scalar function of 3 dimensions. Such *level-set* models have been shown to mimic conventional parametric deformable surface models by encoding surface movements as changes in the greyscale values of a volume dataset. Thus, a well-founded mathematical structure leads to a set of procedures that describes how voxel values can be manipulated to create deformations that are represented as a sequence of volumes. The result is a 3-D morphing method that offers several advantages over previous methods including minimal need for user input, no model parameterization, flexible topology, and sub-voxel accuracy.

Index Terms: Level set method, morphing, solid model, distance function, animation, volume graphics, optimization, deformable model

1 Introduction

Shape metamorphosis is a process where an object continuously changes its own shape into the shape of another object. Within the computer graphics community *morphing* (the vernacular for metamorphosis) has been used frequently for special effects in movies, advertising, and entertainment. *Image morphing* takes a 2-D image of an object and transforms the appearance of that object into the appearance of another object, with the goal of producing natural-appearing, or at least sensible, intermediate images. *Three-dimensional morphing* or *shape morphing* involves smoothly changing the model of one object into the model of another object. Shape morphing algorithms have been developed for both surface models and volumetric models. The surface model algorithms transform the surface patches (usually polygons) of the source model into the surface patches of the target model. The volume-based morphing algorithms represent 3-D objects as volumes and manipulate the voxel values of volumes in order to make one object become another. Volume datasets needed for this process may be acquired directly from 3-D scanning devices, such as MRI or CT, or they may be generated via 3-D scan conversion of solid geometric models.

Despite the increased complexity and computation time associated with morphing 3-D shapes rather than 2-D images, shape morphing does have some distinct advantages. First, image morphing is directly tied to the views of the two input images. Any morphing effect must be based only on the information available in the two images. This prevents the animator from making camera, lighting or shading changes during the morph. If the morph is performed with 3-D techniques, the animator is provided with much more flexibility in presenting the results of the morph. Once a sequence of transforming models has been generated, the animator may experiment with a variety of camera angles and motions for viewing the models. The scene's global lighting and shading parameters may also change during the morph. A second advantage is that a 3-D morph ensures that the resulting rendered images represent continuous sequence of actual 3-D shapes. In image morphing, maintaining the 3-D feasibility of the intermediate images is the responsibility of the user, i.e., care must be taken to make sure intermediate images represent pictures of feasible objects rather than a fuzzy mixture of ghost-like objects.

One disadvantage of 3-D morphing is that it requires two 3-D models. It is not always possible to acquire accurate models of the objects that one wishes to morph, e.g., a sequence of familiar faces. At present, digital photographs of real objects tend to capture more visual detail than 3-D models of the same. Despite

this, there is a class of morphing problems, for which 3-D models exist or are easily obtainable, which lend themselves to the application of shape morphing.

In order to compare the adequacy of different approaches to 3-D shape morphing, we have identified several desirable aspects of a 3-D morphing technology. These properties are:

1. The transition process should begin with a *source* surface and end with a specified *target* surface.
2. Intermediate surfaces should undergo continuous 3-D transitions (rather than continuity only in the image space).
3. The morphing algorithm should apply to a wide range of shapes and topologies.
4. A 3-D morphing algorithm should incorporate user input easily but should degrade gracefully without it.
5. Transitional shapes should depend only on the surface geometry of the two input shapes and user input.

These requirements are not exhaustive, but they capture many of the practical aspects of 3-D morphing.

In this paper we present a new approach to 3-D model morphing. We employ an active deformable surface which starts at the source shape and smoothly changes into the target shape. This deformation process is described by the optimization of an objective function that measures the similarity between the target and the deforming surface. We represent the deformable surface as a level set (iso-surface) of a densely sampled scalar function of 3 dimensions. The sampling produces a regularly-spaced rectilinear volume dataset. Such *level-set* models [37, 41] have been shown to mimic conventional parametric deformable surface models by encoding surface movements as changes in the greyscale values of the volume. A well-founded mathematical structure leads to a set of procedures that describes how voxel values can be manipulated to create deformations in the level sets. The result is a voxel-based modeling technology that offers several advantages over previous methods, including support for a wide range of user input, no need for parameterization, flexible topology, and sub-voxel accuracy. This paper describes the application of this technology to 3-D surface morphing.

This work, as with several other volumetric morphing techniques [11, 26], is based on the image morphing strategy, and consists of two distinct steps. The first step is a *global, geometric warping*, which utilizes a

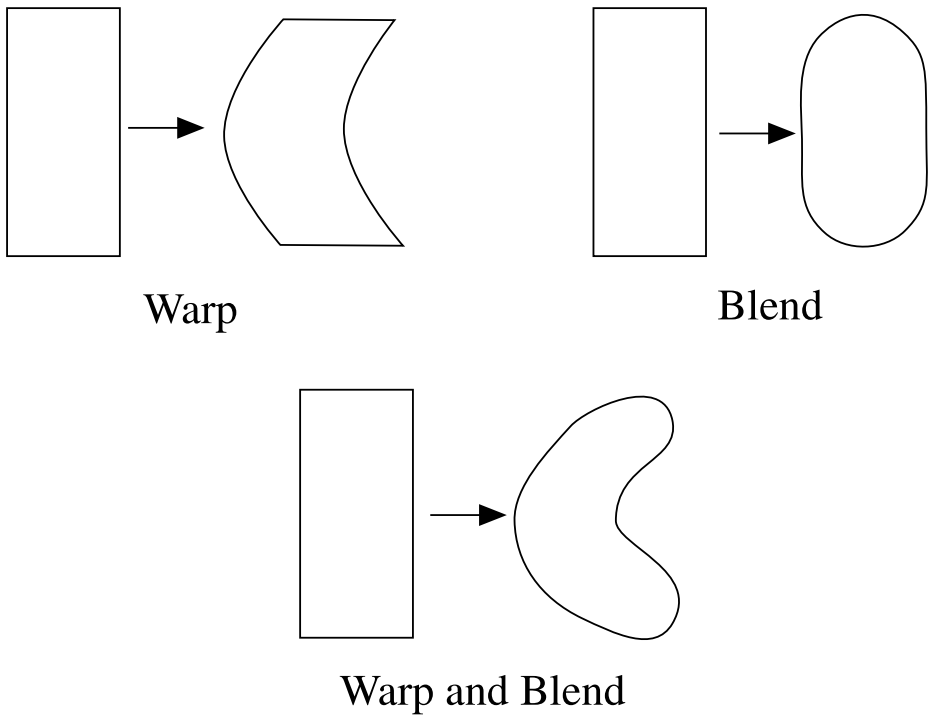


Figure 1: Shape morphing, like image morphing, can be described as a warping combined with a blending.

coordinate transformation that maps the source model into approximately the same shape and orientation as the target model, as shown in Figure 1. This coordinate transformation can take a variety of different forms, but the literature has shown that a rigid transformation combined with a non-rigid, spline-based, coordinate map is quite effective. The non-rigid transformation is usually determined by a set of user-defined correspondences (i.e., fiducials) between the source image/volume and the target. The second step is a *blending* that completes the morph by ensuring that any discrepancies that remain in the images/volumes after the warping are gradually removed over the course of the morphing process. In image morphing and volumetric shape morphing, the blending step is usually achieved by a simple linear interpolation, or cross-dissolve, of the two images/volumes. The warping stage of image/volume morphing has been studied extensively by other researchers, and our work primarily focuses on developing a superior method for the blending stage, one which reduces the amount of user input required to produce an acceptable morphing result. Indeed, the proposed level-set approach to shape metamorphosis will produce a reasonable morph with virtually no user input. Given this technology, user input can be incorporated in an incremental fashion starting with a minimal amount and proceeding with progressively more until a satisfactory result is achieved.

Our morphing approach consists of several stages. First, the source and target objects are defined, with

Constructive Solid Geometry (CSG) models in our examples. The models are scan converted into 3-D distance volumes (a signed distance transform), where the shortest distance to the solid model is stored at each voxel. We use the sign convention that distance is positive inside the object, and negative outside the object. A level-set model is fit to the zero level set of the source model distance volume. A coordinate transformation provides a mapping from every point in the domain of the level-set model into the distance volume of the target model. This is the warping step. The level-set model is then allowed to deform using the signed distance transform of the target model. Each point on the source surface moves in the direction normal to the surface at that point with a velocity proportional to the signed distance transform of the target object at that point in 3-space. This process is the *blending* step of the 3-D surface morph. At user-defined time intervals the level-set model is converted into a polygonal model. This model is expressed in the coordinate system of the source model. Therefore the coordinate transformation is incrementally applied, as a function of time, to the sequence of polygonal models, which are then rendered to produce the frames of the morphing animation.

In general, an animator controls the morph by defining how the models overlap, using a variety of coordinate transformations. This can include rigid transformations, scalings, and non-rigid transformations determined by sets of user-defined fiducials. Our current implementation also provides an automated mechanism for determining rotation, translation, and non-uniform scaling, and thereby offers the *option* of a totally automatic morphing algorithm. The system accomplishes this by overlaying the centroids of the two models, aligning their principal axes, and non-uniformly scaling along these axes. The surface of the level-set model only moves in those areas where the target and the source surfaces are not brought perfectly together by the coordinate transformation. The portion of the level-set model which is outside of the target model will shrink, while the portion inside the target model will expand to become the final shape.

The level-set approach to 3-D model morphing provides several advantages over other 3-D morphing algorithms. Level-set models are active; their underlying motion is defined algorithmically rather than interactively, guaranteeing that the source shape will become the target shape, given that both models initially overlap. The amount of user input required to produce a morph is directly proportional to the amount of control the animator wishes to impose on the process. The animator may allow the system to automatically generate the morph, or he/she can employ a full 3-D warping to describe the morph, with the level sets simply providing a small fine-tuning of the surface. Any intermediate amount of user input will produce a

reasonable morph. Because level-set models do not rely on any kind of parameterization, they do not suffer the problems of parametric surfaces, e.g. a limited set of possible shapes and the need for reparameterization after undergoing significant changes in shape. This lack of parameterization, along with no direct representation of topological structure, allow level-set models to easily change topology while morphing. The model can "split" into pieces to form multiple objects. Conversely several disjoint objects may come together to make a single object. Finally, both the scan conversion and deformation stages of our morphing approach produce models with user-defined sub-voxel accuracy. This provides a flexible time-quality trade off and generates superior results at low volume resolutions as compared to previous work.

The remainder of the paper proceeds as follows. The next section describes related morphing work. The section following that describes the novel aspect of our method—shape interpolation based on the optimization of a similarity function using level-set models. It is followed by a discussion of general properties and numerical methods. Section 6 describes all stages of our morphing approach. The paper closes with three morphing examples, conclusions, and some thoughts on future work.

2 Related Work

In recent years numerous 3-D morphing algorithms have been reported in the literature. These algorithms generally fall into two categories, surface-based approaches and volume-based approaches. The former primarily consists of those methods that continuously change one polygonal surface into another. Volumetric methods modify the voxel values of a volume dataset in order to smoothly transform the object defined by the volume from a source shape into a target shape. A third class of algorithms involves morphing implicit models, either by transforming the underlying structure of "blobby" models [14], or by creating higher dimensional interpolating implicit functions [39].

Kent et al. [23] described the fundamentals for morphing 3-D polygonal surfaces. The first step is to create a single topological description of vertices, edges and faces, which contains the combined topological structure of both the source and target surfaces (the correspondence phase). The vertices of the structure are then interpolated between their position on the initial surface to their position on the target surface to create the morphing result. Parent [31] presents an improved technique for establishing vertex and edge correspondences when creating the combined topological structure. Chen et al. [9] applied the same approach to polygonal

surfaces defined in cylindrical coordinates. Kanai et al. [21] use harmonic maps to define the correspondences in the combined topological structure. Lazarus and Verroust [24] do not build a common adjacency graph, but instead create a common parameterized mesh during a sampling process. Gregory et al. [17] present a method to assist the user in defining these correspondences. Lee et al. [25] apply many of these surface-based methods to morphing multiresolution meshes. Rossignac and Kaul [22, 34] introduce the notion of an interpolating polyhedron, an application of mathematical morphology to face sets. They also describe a user control methodology based on a Bezier curve paradigm. DeCarlo and Gallier [12] demonstrate that it is possible (with significant effort) to morph polygonal models of differing genus. Surface-based methods are important because of the wealth of polygonal models available to animators, but they are burdened with the difficult task of creating a single topological structure which can represent the source and target surfaces. While it has been shown that changing the genus of a morphing polygonal surface is possible, for example from a sphere to a torus, it can be a difficult and tedious process which requires significant user input. Another troubling feature of surface-based methods is the problem of self-intersection. These methods cannot guarantee that polygonal surfaces will not pass through themselves, creating physically nonsensical intermediate results.

Embedding the morphing surfaces in volumes alleviates these problems. Hughes [19] demonstrates how volumes can be used to create a morph between objects of different genus. The approach involves linearly interpolating the Fourier transforms of the volumes. A schedule is used during interpolation which filters out the high frequencies of the initial volume, while interpolating the low frequencies of both volumes, and gradually adds the high frequencies of the target volume. He et al. [18] propose a similar approach using wavelets to control the high-frequency artifacts. They also developed methods that allow the user some control of the interpolation by establishing explicit correspondences between the volumes. Lerios et al. [26] describe a morphing method which is a 3-D extension of Beier and Neely's [4] 2-D (image) morphing technique. The first step is to apply a user-defined geometric warp to the source object in order to deform it into approximately the same shape as the target object. This step allows the user to specify corresponding features on the two objects. The second step interpolates the values between matching voxels in the (warped) source and target volumes. Chen et al. [10] propose disk fields as a superior instrument for specifying the warps needed for this kind of 3-D morphing. Payne and Toga [32] describe a method for changing one volumetric model into another by interpolating the distance fields generated from the two volumes. A distance field, or distance volume, is a volume dataset where the value stored at each voxel is the shortest

distance to the surface of the object being represented by the volume. Cohen-Or et al. [11] improve upon this approach by including a two-part warping step that calculates a rigid transformation and an elastic warping based upon user-supplied anchor points. This extra step approximately aligns the target and final objects, and provides significant user control to the overall morphing process.

The proposed volume-based morphing technique shares many of the advantages of previous volume-based methods. It can easily morph objects of different genus. It is not burdened with the difficult vertex/edge/face book-keeping of surface methods, and surfaces cannot pass through each other. Additionally, our method provides a novel blending/interpolation mechanism that will not produce the ghosting or spontaneous generation of new objects which is possible when simply interpolating voxel intensity or distance values (for example, see Figures 4, 5 and 6). Our method does not require user input to produce a reasonable morph, but can easily incorporate previously published warping techniques to provide a wide range of animator control. Finally, because we calculate the distance transform to sub-voxel accuracy, and effectively track a deforming surface within voxels our results do not produce the aliasing artifacts commonly found in other distance-based, volumetric approaches.

3 Metamorphosis As A Goal-Driven Process

Morphing strategies may be built upon any number of underlying principles. For instance, surface metamorphosis could consist of a sequence of time slices from a four-dimensional manifold which optimizes some space-time criteria [39]. The proposed strategy for object metamorphosis is based on yet another principle: shape metamorphosis is the process by which one object seeks to *resemble* another. This philosophy raises two questions. First, what is the metric by which we can quantify the similarity of two objects? Second, what is the process by which one object seeks to optimize that metric? This paper shows that even very simple answers to these two questions produce very powerful algorithms for shape metamorphosis, with a great deal of opportunity for future enhancement and refinement of the resulting algorithms.

In order to create a very general algorithm for metamorphosis, we work with a very general notion of a surface. Consider an open set $\Omega_A \subset \mathbb{R}^3$, which is the source object, and a target $\Omega_B \subset \mathbb{R}^3$. The source, Ω_A, is enclosed by a surface $\mathcal{S}_A = \partial\Omega_A$, and likewise $\mathcal{S}_B = \partial\Omega_B$. We require that the objects be compact and lie in some finite domain $U \subset \mathbb{R}^3$. Notice that we do not require any specific connectivity or topology,

which means that each object could consist of a set of disconnected pieces (each with any number of holes) all sitting in U.

We propose a very simple metric for comparing two shapes that maximizes the volume shared by the interiors of the two objects. First define an inside-outside function for the target, $\gamma_B : \mathbb{R}^3 \mapsto \mathbb{R}$ for Ω_B, such that

$$\begin{aligned} \gamma_B(\mathbf{x}) &= 0 \quad \forall\, \mathbf{x} \in \mathcal{S}_B \\ \gamma_B(\mathbf{x}) &> 0 \quad \forall\, \mathbf{x} \in \Omega_B \\ \gamma_B(\mathbf{x}) &< 0 \quad \text{otherwise} \ . \end{aligned} \tag{1}$$

The inside-outside function γ_B can be used to quantify the extent to which an intermediate object Ω_t overlaps with the target, Ω_B, with the volume integral

$$\mathcal{M}_{\Omega_B,\Omega_t} = \int_{\Omega_t} \gamma_B(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{2}$$

Notice, this integral achieves its maximum,

$$\Omega_m = \arg\max_{\Omega_t} \left(\int_{\Omega_t} \gamma_B(\mathbf{x}) \mathrm{d}\mathbf{x} \right) \tag{3}$$

when $\Omega_m = \Omega_B$, because in that case the integral includes all of the positive parts of γ_B and none of the negative parts.

We can compute the first variation of this metric with respect to Ω_t by noting that incremental changes in the object shape can be expressed in terms of the surface that encloses it:

$$\int_{\Omega_t + \mathrm{d}\Omega_t} \gamma_B(\mathbf{x}) \mathrm{d}\mathbf{x} = \int_{\Omega_t} \gamma_B(\mathbf{x}) \mathrm{d}\mathbf{x} + \int_{\mathcal{S}_t} \gamma_B(\mathbf{x}) \epsilon(\mathbf{x}) \cdot \mathbf{N}(\mathbf{x}) \mathrm{d}\mathbf{x}, \tag{4}$$

where $\mathbf{N} : \mathcal{S}_t \mapsto S^3$ is the surface normal, which is a mapping from every point on the surface to the unit sphere. Differentiating with respect to ϵ gives the first variation [13] with respect to surface position:

$$\mathrm{d}\mathcal{M} = \gamma_B(\mathbf{x}) \mathbf{N}(\mathbf{x}). \tag{5}$$

Using the first variation from Equation 5 and a hill climbing strategy, we obtain the surface motion, defined for each surface point, that minimizes our similarity metric:

$$\frac{\mathrm{d}\mathbf{s}}{\mathrm{d}t} = \gamma_B\left(\mathbf{s}(t)\right) \mathbf{N}\left(\mathbf{s}(t)\right) \ \forall\, \mathbf{s}(t) \ \in \ \mathcal{S}_t. \tag{6}$$

This equation states that each point on the surface $\mathcal{S}_t$ moves at each time step $\mathrm{d}t$ in the direction of the

surface normal $\mathbf{N}(\mathbf{s}(t))$ with a step size proportional to the value of the value of the inside-outside function $\gamma_B(\mathbf{s}(t))$ at the point location.

4 Properties

This section describes the properties of the solutions of the partial differential equation given in (6). For this paper, we examine the case of simple, closed, surfaces, but the results easily generalize to more complicated surfaces. In this section we systematically show that if the initial and target objects overlap, the final solution of the metamorphosis will be identical to the target.

4.1 Steady-State Solution

We begin by examining properties of the steady-state solution of Equation 6, which is the surface $\mathcal{S}_t$ such that $\partial \mathbf{s}(t)/\partial t = 0$ for all $\mathbf{s} \in \mathcal{S}_t$. Thus, for steady state, which we denote $\mathcal{S}_\infty$,

$$\gamma_B(\mathbf{x})\mathbf{N}(\mathbf{s}) = 0 \Rightarrow \gamma_B(\mathbf{s}) = 0 \ \ \forall \mathbf{s} \in \mathcal{S}_\infty. \tag{7}$$

The inside-outside function for the target, γ_B, has the property that

$$\gamma_B(\mathbf{x}) = 0 \ \ \Leftrightarrow \ \ \mathbf{x} \ \in \ \mathcal{S}_B. \tag{8}$$

Thus, we can conclude $\mathcal{S}_\infty \subseteq \mathcal{S}_B$. From this it follows that if $\mathcal{S}_\infty$ is compact and closed, it must have one of two forms:

1. $\mathcal{S}_\infty = \emptyset$,
2. $\mathcal{S}_\infty = \mathcal{S}_B$.

The first case is trivial, because there are no points on $\mathcal{S}_\infty$ to violate the condition given in (7). The second case follows from the fact that simple, closed, connected surfaces have no proper subsets that are also closed surfaces. Thus

$$\mathcal{S}_\infty \subseteq \mathcal{S}_B \ \text{ and } \ \mathcal{S}_\infty \ \text{ closed} \quad \Rightarrow \ \ \mathcal{S}_\infty = \mathcal{S}_\mathrm{B}. \tag{9}$$

Also, because the first variation of the metric $\mathcal{M}_{\Omega_B,\Omega_t}$ is zero only where $\mathcal{S}_t = \mathcal{S}_B$, it achieves its maximum for that solution with $\Omega_t = \Omega_B$.

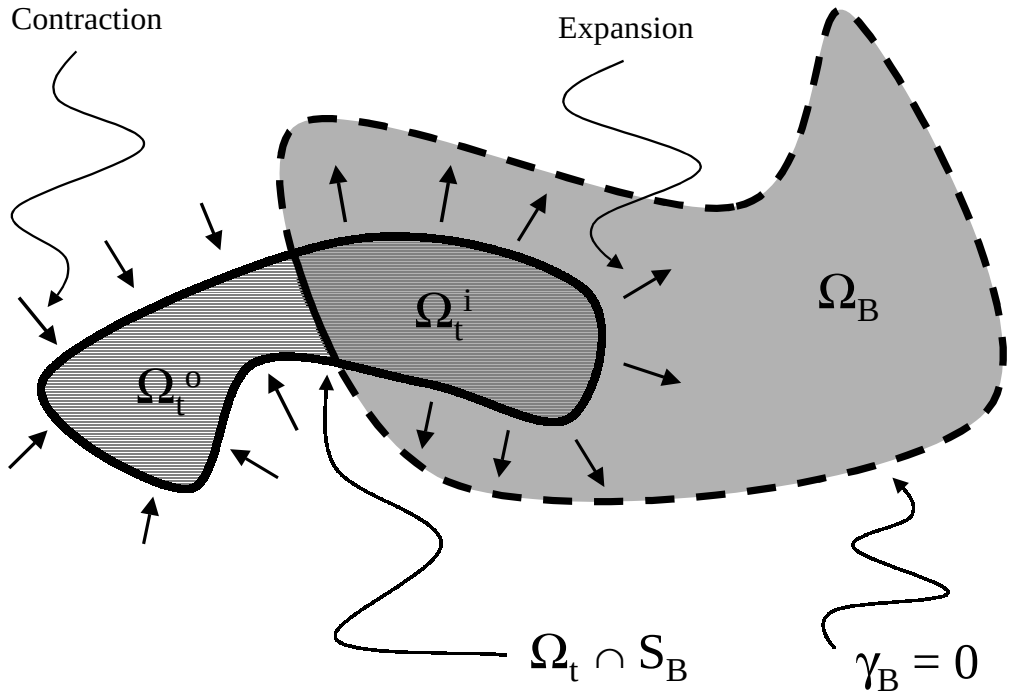


Figure 2: In order to see that overlapping models converge we can divide the model Ω_t into two parts separated by a stationary boundary; one part contracts to annihilation while the other expands until it is equal to the target.

Another important aspect of the algorithm is that a component of the source that overlaps with a component of the target will deform in such a way that it takes on the shape of that target component.[1] This follows from the fact the metric $\mathcal{M}_{\Omega_B,\Omega_t}$ can be decomposed into two parts (as in Figure 2):

$$\mathcal{M}_{\Omega_B,\Omega_t} = \mathcal{M}^i_{\Omega_B,\Omega_t} + \mathcal{M}^o_{\Omega_B,\Omega_t} = \int_{\Omega_t^i} \gamma_B(\mathbf{x}) + \int_{\Omega_t^o} \gamma_B(\mathbf{x}), \tag{10}$$

where $\Omega_t^i = \Omega_t \cap \Omega_B$ and $\Omega_t^o = \Omega_t - \Omega_t^i - \mathcal{S}_B$. In this case the first variation produces a pair of deformable surfaces that share a boundary along $\Omega_t \cap \mathcal{S}_B$. The exterior model remains on the exterior of $\mathcal{S}_B$ and contracts (increasing the value of $\mathcal{M}^o$) until it collapses into itself along $\mathcal{S}_B$ and the value of $\mathcal{M}^o$ reaches zero. The interior model expands, cannot pass into the exterior of $\mathcal{S}_B$, and increases the value of $\mathcal{M}^i$ at a rate

$$\frac{\partial \mathcal{M}^i}{\partial t} = \int_{\mathcal{S}_t^i} \gamma_B(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{11}$$

[1] A target object may consist of several, disjoint components.

Because $\gamma_B(\mathbf{x})$ is nonnegative on the interior of the target, Ω_B, the metric does not stop increasing (and thus the surface keeps moving) until the surface reaches its steady state, i.e., $\gamma_B(\mathbf{x}) = 0 \ \forall \ \mathbf{x} \in \mathcal{S}_t^i$. Thus, if a source component overlaps with a target component, the resulting morph does not terminate until it reaches the shape of the target.

4.2 Strategy

This formulation suggests a strategy for shape metamorphosis: construct a family of objects Ω_t (with a corresponding $\mathcal{S}_t$), with $\Omega_t|_{t=0} = \Omega_A$ that evolve according to a hill-climbing strategy, and maximize $\mathcal{M}$. If properly initialized (i.e., all of the components of Ω_B have some overlap with Ω_A) then the deformation will seek the global maximum, which is the target Ω_B.

The intermediate shapes contained in Ω_t depend, of course, on the choice of γ_B. In order to avoid numerical difficulties and to avoid discontinuities in the solution, γ_B should be continuous.[2] Furthermore γ_B should reward shapes that are similar to the target but offset by some small distance. That is γ_B should carry information about the shape of the surface into 3-D so that shapes tend to "look like" the target as they get nearer. This suggests that a natural choice of γ_B is the *signed distance transform* [2, 5] of the target surface $\mathcal{S}_B$ or some monotonic function thereof. Thus, if we let $D_B : U \mapsto \mathbb{R}$ be the signed distance transform, $\gamma_B(\mathbf{x}) = f\left(D_B(\mathbf{x})\right)$, where $f(0) = 0$ and $f'(a) > 0$. By tuning f one can control the way the metamorphosis behaves when intermediate surfaces are far from the target. If $f(a) = a$, then $\mathcal{S}_t$ contracts or expands with a magnitude that depends on the signed distance to the target.

As it stands this process is a "low-level" approach to shape metamorphosis, which can form continuous deformations for shapes that are somehow "close" in their initial shapes. As described in the introduction, this low-level process is meant to address the blending stage of 3-D metamorphosis—it is meant to be combined with a higher-level process which accounts for "semantic" aspects of shape correspondence. Incorporating user input is important for any shape morphing technique, because in many cases finding the best set of transition shapes depends on context. Only users can apply semantic considerations to the transformation of one object to another. Our assertion, however, is that this underlying coordinate transformation can achieve only some finite similarity between the "warped" source model and the target, and even this may

[2] In general signed distance transforms are C0 continuous, but not C1 continuous.

require a great deal of user input. In the event that a user is not able or willing to define every important correspondence between two objects, some other method must "fill in" the gaps remaining between the source and target surface. We propose the use of deformable level-set models, to achieve a continuous transition (blending) between the shapes that result from the underlying coordinate transformation.

This higher-level information affects the blending process through a 3-D coordinate transformation, T. T maps a point $\mathbf{x}$ in the coordinate system of the source Ω_A into the coordinate system of the target Ω_B. As described in the introduction, this transformation is meant to be quite general; it can accommodate a wide range of *global* deformations. However, if the blending is sufficiently powerful and can adequately deal with significant shape discrepancies (after the coordinate transformation), only a crude alignment of the source and target is necessary. The coordinate transformation enters into the deformation process through the term γ_B, which quantifies the proximity of one surface to another by means of the distance transform. If we express the distance transform for the target in coordinates of the target, then we have $\gamma_B(\mathbf{x}) = f\left(D_B\left(T(\mathbf{x})\right)\right)$. Alternatively one could resample the distance transform of Ω_B in the coordinate system of Ω_A, i.e., $\gamma'_B(\mathbf{x}) = \gamma_B\left(T(\mathbf{x})\right)$.

5 Level-Set Models

In practice, the strategy of shape metamorphosis by surface deformations described in the previous section must be computed using some specific surface representation. Ideally, we would like the surface representation to be as general as the underlying theory. For this we use the method of *level-set models* [37], which is a technique for modeling surfaces as iso-values of a densely sampled scalar function over the domain U. Surface movements are encoded as changes in the greyscale values of the voxels. Using level-set models, we can compute goal-driven deformations on surfaces without any explicit surface representation.

5.1 Theory

The strategy of level-set models represents a set of surface points $\mathcal{S}$ as an iso-surface of ϕ, which we will call the embedding.

$$\mathcal{S} = \{\mathbf{x} | \phi(\mathbf{x}) = k\}, \tag{12}$$

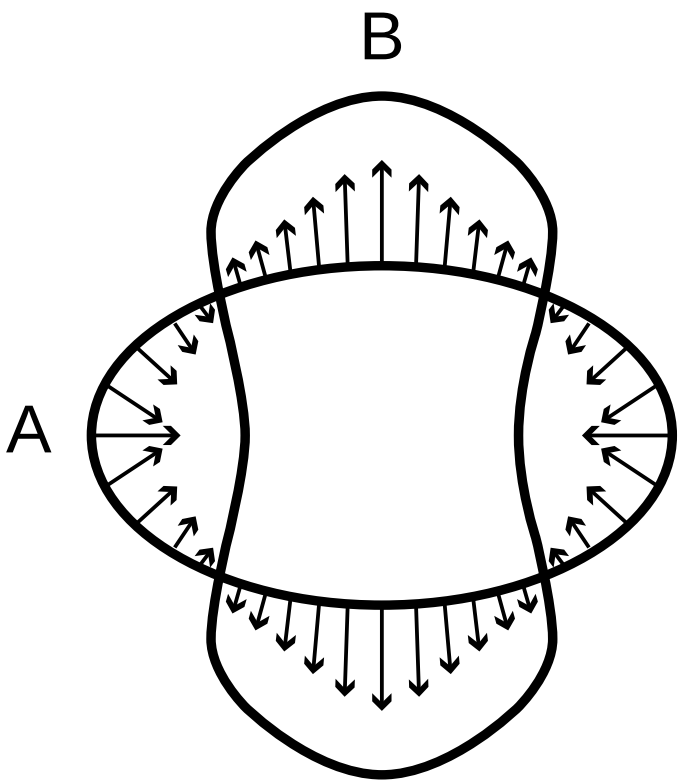


Figure 3: A 2-D morphing example showing scaled surface normals.

where the value of k is arbitrary and will fall out in subsequent calculations.[3] We can also represent a family of surfaces and a corresponding family of embeddings:

$$\mathcal{S}_t = \{\mathbf{x} | \phi(\mathbf{x}, t) = k\}. \tag{13}$$

Consider a point $\mathbf{s}(t)$ on the surface that moves through space as a function of t. Because $\mathbf{s}(t)$ remains the kth level-surface of ϕ over time, the total derivative of ϕ with respect to time must be zero. Thus,

$$\frac{\partial \phi(\mathbf{s}(t), t)}{\partial t} + \nabla \phi(\mathbf{s}(t), t) \cdot \frac{\mathrm{d}\mathbf{s}(t)}{\mathrm{d}t} = 0, \tag{14}$$

which establishes the connection between the way points on $\mathcal{S}_t$ move and the way the greyscale values (at positions on the surface) of the embedding change. We can rewrite Equation 14 as follows:

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\mathbf{s}(t)}{\mathrm{d}t} = |\nabla \phi| \frac{\mathrm{d}\mathbf{s}(t)}{\mathrm{d}t} \cdot \mathbf{N}(\mathbf{s}), \tag{15}$$

using the fact that $\nabla \phi = |\nabla \phi| \mathbf{N}$.

[3] For the remainder of this paper we use the convention that $k = 0$ is the level set of interest.

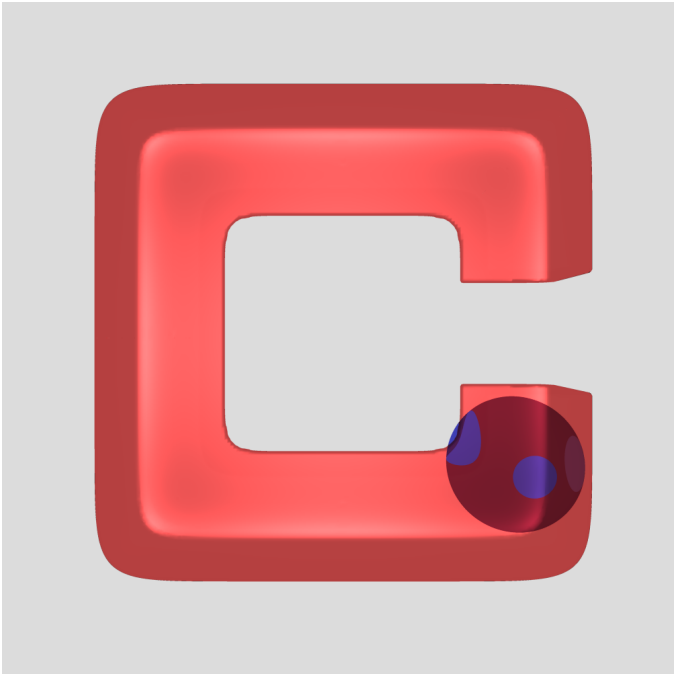


Figure 4: Initial configuration for a morphing sequence.

We can now "plug" into Equation 15 any surface motion we wish to compute. Thus, inserting $\mathrm{d}\mathbf{s}(t)/\mathrm{d}t$ from Equation 6, which describes the motion of the surface model as it becomes more like the target, yields

$$\frac{\partial \phi(\mathbf{s})}{\partial t} = |\nabla \phi| \gamma_B \left(\mathbf{s}(t)\right), \tag{16}$$

where we have used the fact that surface normal is unit length, i.e. $N \cdot N = 1$. Notice, we have not chosen a particular k, and therefore this analysis applies to *every* level set of ϕ. Thus we are describing the deformation of an embedded family of surface models each of which evolves according to the same equation:

$$\frac{\partial \phi(\mathbf{x})}{\partial t} = |\nabla \phi(\mathbf{x})| \gamma_B(\mathbf{x}). \tag{17}$$

The particular level set of interest is the one that we choose, by construction of the initial conditions, such that $\phi(\mathbf{x}, 0) = k \ \forall \ \mathbf{x} \in \mathcal{S}_A$.

The deformation process, defined by Equations 6 and 17, is further detailed in Figure 3 with a two-dimensional example. Given the inputs Ω_A and Ω_B, surface $\mathcal{S}_t$ will begin on the surface of $\mathcal{S}_A$. Each point on the surface will move in the direction of the normal at that point with a velocity proportional to the signed distance at that point in 3-space from $\mathcal{S}_B$. Those parts of $\mathcal{S}_t$ that are outside of Ω_B will contract, because D_B is negative in those regions. The parts of $\mathcal{S}_t$ inside of Ω_B will move in the direction of the surface normal, and

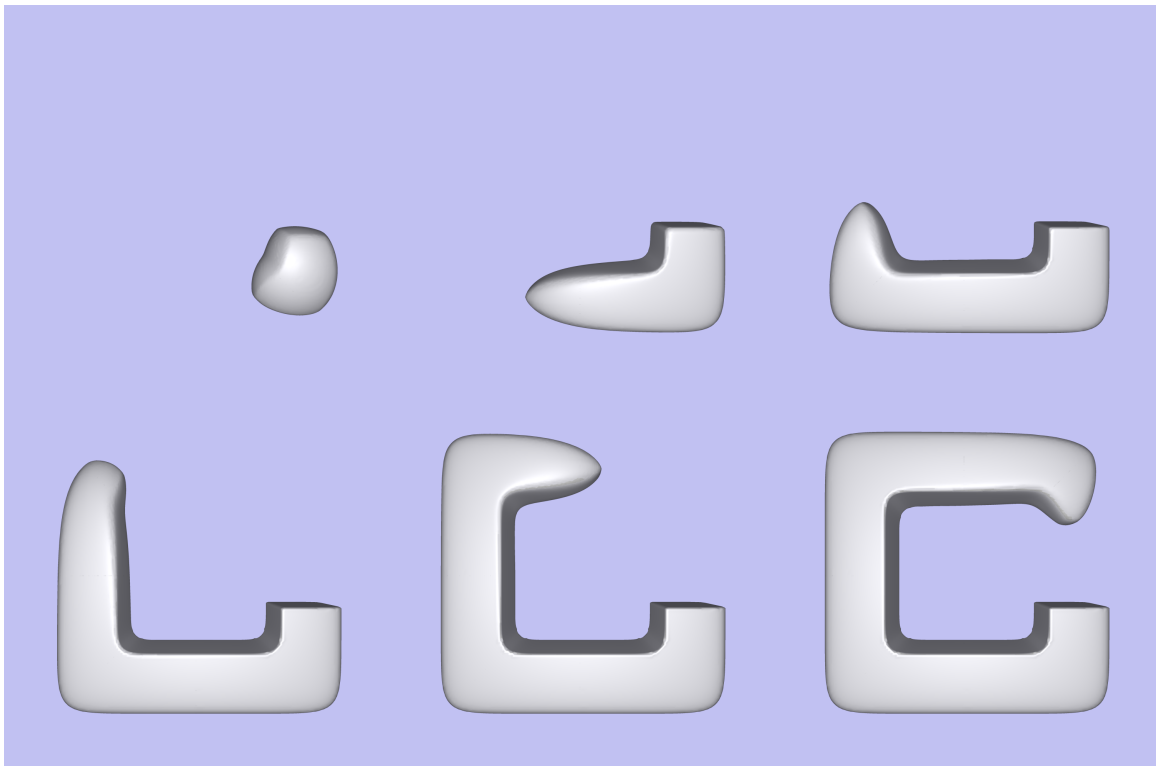


Figure 5: A 3-D level-set morphing example.

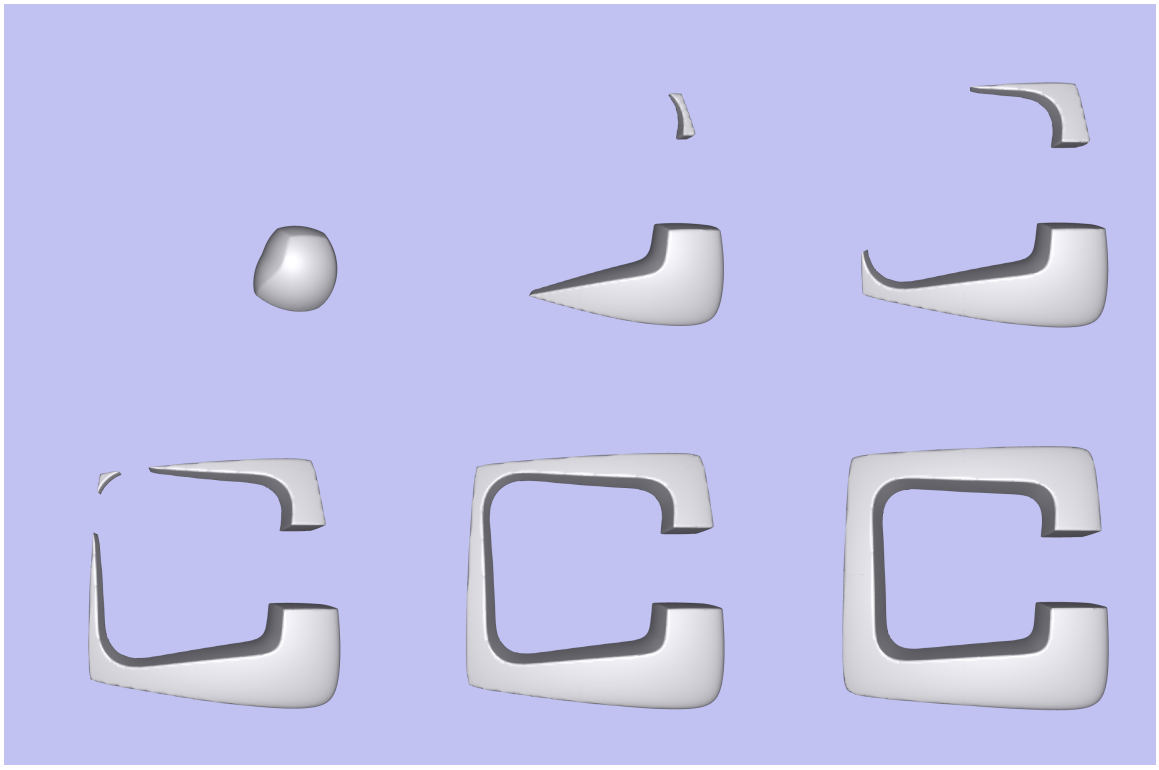


Figure 6: Interpolation of two distance volumes.

will expand. Segments of the surface further away from $\mathcal{S}_B$ will move faster towards $\mathcal{S}_B$ than the segments closer to the surface. As segments of $\mathcal{S}_t$ reach the surface of Ω_B they will no longer move because D_B goes to zero in these regions.

A three-dimensional morphing example is presented in Figures 4 and 5. Figure 4 presents the initial configuration for the morphing sequence. A small sphere will morph into the larger "C" shape. The two objects are first rendered as translucent objects to give some indication of their initial overlap. The ball is mostly contained inside the "C" shape, with the light blue regions of the sphere protruding outside the red "C". The morphing sequence in Figure 5 demonstrates that the regions outside of the "C" contract slightly to fit to the surface of the "C". The surface regions of the sphere inside of the "C" expand to fill the remainder of the object. This also demonstrates that having a surface expand in the direction of its local normal will allow it to deform to fit concave objects, a fact also shown by Miller et al. [29]. Figure 6 contrasts our method of blending with the method used in several other volume-based morphing techniques. Here, the voxel values of the two initial distance volumes are simply interpolated. It can be seen that voxel interpolation produces undesirable artifacts, namely pieces of the "C" shape "pop out of thin air." This is a typical problem when using voxel interpolation to blend volumetric objects. In order to overcome this problem in this example, the sphere would have to be geometrically warped into approximately the same shape as the "C". Our method requires no warping. The user specifies the initial overlap of the two objects and the level-set deformation completes the morph.

The complete deformation strategy is as follows. First, initialize a volume so that the kth level set is, approximately, aligned with $\mathcal{S}_B$. For all of our work we will use the zero-set as the level-set model. This initialization can be done by using the discrete distance transform of $\mathcal{S}_B$, which we compute from CSG models using the method of Breen et al. [7, 8]. We use this initialization to solve the initial value problem given by Equation 17, using the distance transform of the target as the γ_B. We solve this equation using finite forward differences, as described in the next section. When the model is sufficiently close to the target (a threshold on the RMS distance to the target), the process stops and the metamorphosis is complete.

5.2 Implementation

5.2.1 Numerical Issues

The solutions to the partial differential equations described in Section 5.1 are computed using finite differences on a discrete grid[4] The use of a grid and discrete time steps raises a number of numerical and computational issues that are important to the implementation.

The first issue is the discrete approximation of the derivatives in Equation 17. Let u^n be a discrete approximation to $\phi(\mathbf{x}, t)$ at the nth discrete time step. The equation can be solved using finite forward differences if one uses the up-wind scheme, proposed by Osher and Sethian [30], to compute the spatial derivatives. The update equation is

$$u^{n+1}_{i,j,k} = u^n_{i,j,k} + \Delta t \Delta u^n_{i,j,k}, \tag{18}$$

where Δt is a constant that is chosen to ensure stability, and $\Delta u^n_{i,j,k}$ is the discrete approximation to $\partial\phi/\partial t$. We assume, without a loss in generality, that the grid spacing is unity. The initial conditions u^0 are established by the algorithm and the boundary conditions are such that the derivatives toward the outside of the grid are zero (Neumann type).

The up-wind scheme relies on *one-side* derivatives:

$$\delta^+_x u^n_{i,j,k} = u^n_{i+1,j,k} - u^n_{i,j,k}, \tag{19}$$

$$\delta^-_x u^n_{i,j,k} = u^n_{i,j,k} - u^n_{i-1,j,k}, \tag{20}$$

$$\delta^+_y u^n_{i,j,k} = u^n_{i,j+1,k} - u^n_{i,j,k}, \tag{21}$$

$$\delta^-_y u^n_{i,j,k} = u^n_{i,j,k} - u^n_{i,j-1,k}, \tag{22}$$

and so forth. The partials in Equation 17 are computed using only those derivatives that are up-wind relative to the movement of the level set. Thus, the update becomes

$$\Delta u_{i,j,k} = \gamma_B(i,j,k) \cdot \begin{cases} \left(\sum_{w\in x,y,z} \min(\delta^+_w u^n_{i,j,k}, 0)^2 + \sum_{w\in x,y,z} \max(\delta^-_w u^n_{i,j,k}, 0)^2\right)^{\frac{1}{2}} & \text{for } \gamma_{\mathrm{B}}(\mathrm{i,j,k}) \geq 0 \\ \left(\sum_{w\in x,y,z} \max(\delta^+_w u^n_{i,j,k}, 0)^2 + \sum_{w\in x,y,z} \min(\delta^-_w u^n_{i,j,k}, 0)^2\right)^{\frac{1}{2}} & \text{for } \gamma_{\mathrm{B}}(\mathrm{i,j,k}) < 0. \end{cases} \tag{23}$$

[4]The level-set software used to produce the morphing results in this paper is available for public use. Contact Ross Whitaker at whitaker@cs.utah.edu for more information.

The time steps, Δt, are limited by the speed of the fastest moving wavefront, which can move only one grid unit per iteration, i.e.,

$$\Delta t \leq \frac{1}{3 \max_{i,j,k} |\gamma_B(i,j,k)|}. \tag{24}$$

In practice, for the purposes of computer animation, one might further limit the time steps to obtain sequences with a sufficient number of in-between frames.

Thus, the level-set method for computing the 3D shape metamorphosis is as follows:

1. Initialize model volume u^0 by sampling the inside-outside function of the source.
2. Construct the volume v, by sampling the inside-outside function of the target.
3. $v_{\max} = 0$
4. For each voxel (i, j, k):
   (a) Find $v_{i,j,k}$.
   (b) $v_{\max} = MAX(|v_{i,j,k}|, v_{\max})$
   (c) Calculate derivatives and the total change at (i, j, k) using nearest neighbors according to the up-wind scheme given in Equation 23.
   (d) Save $\Delta u^n_{i,j,k}$ in a separate volume.
5. Compute Δt according to Equation 24.
6. For each voxel (i, j, k):
   (a) Update $u^{n+1}_{i,j,k}$ according to Equation 18
   (b) Compute the stopping criterion, either RMS change or RMS distance to target.
   (c) If stopping criterion is met, finish, otherwise go to step 4

5.2.2 Sparse-Field Solutions

The up-wind solutions to the equations described in the previous section produces the motion of level-set models over the entire range of the embedding, i.e., for all values of k in Equation 13. However, this method

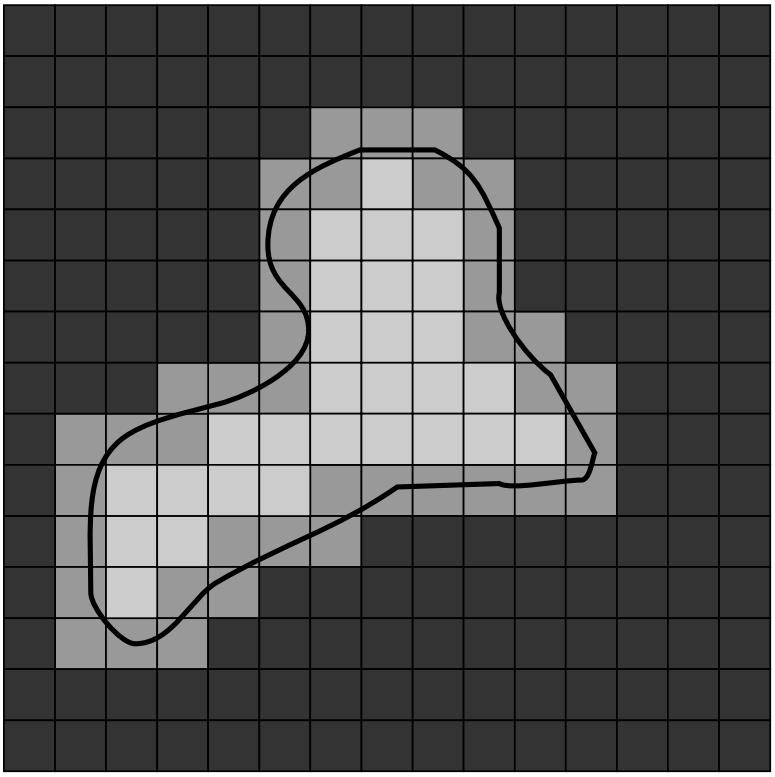


Figure 7: A level curve of a 2-D scalar field passes through a finite set of grid points. Only those grid points and their nearest neighbors are relevant to the evolution of that curve.

requires updating *every voxel in the volume for each iteration.*, which means that the computation time increases as a function of the volume, rather than the surface area, of the model. Because the application of this paper, surface metamorphosis, requires only a single model, the calculation of solutions over the entire range of iso-values is an unnecessary computational burden.

The literature has shown this situation can be improved by the use of *narrow-band* methods, which compute solutions only in a narrow band of voxels that surround the level set of interest [1, 28]. In previous work [40, 42] we described an alternative numerical algorithm, called the sparse-field method, that computes the geometry of only a small subset of points in the range and requires a fraction of the computation time required by previous algorithms. We have shown two advantages to this method. The first is a significant improvement in computation times. The second is increased accuracy when fitting models to forcing functions that are defined to sub-voxel accuracy.

The sparse-field algorithm takes advantage of the fact that a k-level surface, S, of a discrete image u (of any dimension) has a set of cells through which it passes, as shown in Figure 7. The set of grid points adjacent

to the level set is called the *active set*, and the individual elements of this set are called *active points*. As a first-order approximation, the distance of the level set from the center of any active point is proportional to the value of u divided by the gradient magnitude at that point. We compute the evolution given by Equation 17 on the active set and then update the neighborhood around the active set using a fast approximation to the distance transform, which simply adds the "city-block" distance to values of the active set. Because active points must be adjacent to the level-set model, their positions lie within a fixed distance to the model. Therefore the values of u for elements in the active set must lie within a certain range of greyscale values. When active-point values move out of this *active range* they are no longer adjacent to the model. They must be removed from the set and other grid points, those whose values are moving into the active range, must be added to take their place. The precise ordering and execution of these operations is important to the operation of the algorithm.

The values of the points in the active set can be updated using the up-wind scheme described in the previous section. In order to maintain stability, one must update the neighborhoods of active grid points in a way that allows grid points to enter and leave the active set without those changes in status affecting their values. Grid points should be removed from the active set when they are no longer the nearest grid point to the zero crossing. If we assume that the embedding u is a discrete approximation to the distance transform of the model, then the distance of a particular grid point, (i, j, k), to the level set is given by the value of u at that grid point. If the distance between grid points is defined to be unity, then we should remove a point from the active set when the value of u at that point no longer lies in the interval $[-\frac{1}{2}, \frac{1}{2}]$. If the neighbors of that point maintain their distance of 1, then those neighbors will move into the active range just as (i, j, k) is ready to be removed.

There are two operations that are significant to the evolution of the active set. First, the values of u at active points change from one iteration to the next. Second, as the values of active points pass out of the active range they are removed from the active set and other neighboring grid points are added to the active set to take their place. Formal definitions of active sets and the operations that affect them are detailed in [42], and it is shown that active sets will always form a boundary between positive and negative regions in the image, even as control of the level set passes from one set of active points to another.

Because grid points that are near the active set are kept at a fixed value difference from the active points,

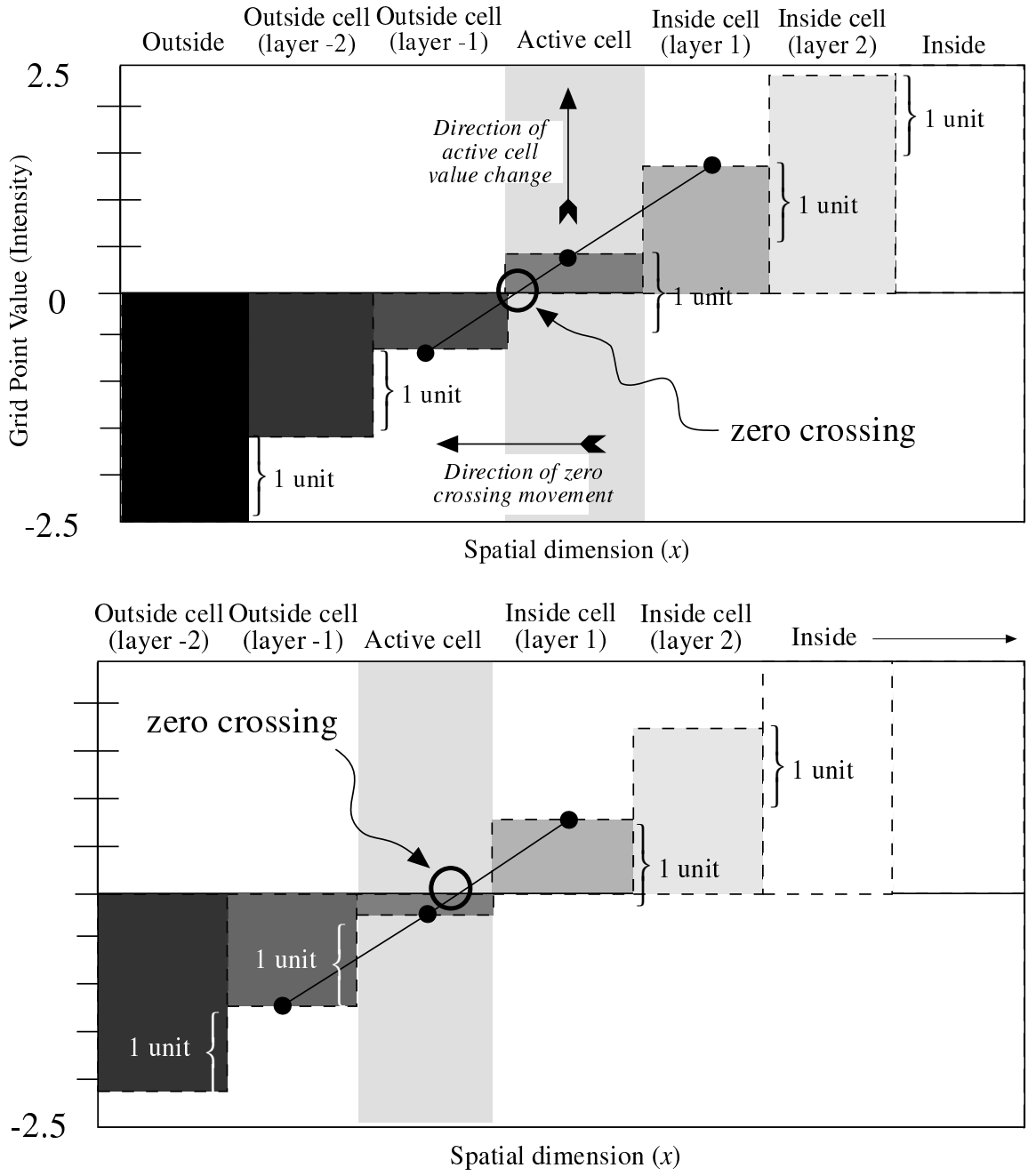


Figure 8: The status of grid points and their values at two different points in time show that as the zero crossing moves, *activity* is passed one grid point to another.

active points serve to control the behavior of adjacent nonactive grid points. The neighborhoods of the active set are defined in *layers*, $L_{+1}, \ldots, L_{\ell}, \ldots, L_{+N}$ and $L_{-1}, \ldots, L_{-\ell}, \ldots, L_{-N}$, where the ℓ indicates the distance (city block distance) from the nearest active grid point, and negative numbers are used for the outside layers. For notational convenience the active set is denoted L_0. The number of layers should coincide with the size of the footprint or neighborhood used to calculate derivatives. In this way, the inside and outside grid points undergo no changes in their values that affect or distort the evolution of the zero set. The work in this paper uses only first-order derivatives of ϕ, which are calculated using nearest neighbors (6 connected). Therefore only 3 layers are necessary (1 inside layer, 1 outside layer, and the active set). These layers are denoted L_1, L_{-1}, and L_0. The active set has grid point values in the range $[-\frac{1}{2}, \frac{1}{2}]$. The values of the grid points in each neighborhood layer are kept 1 unit from the next layer closest to the active set as shown in Figure 8. Thus the values of layer L_ℓ fall in the interval $[\ell - \frac{1}{2}, \ell + \frac{1}{2}]$. For $2N + 1$ layers, the values of the grid points that are totally inside and outside are $N + \frac{1}{2}$ and $-N - \frac{1}{2}$, respectively.

This algorithm can be implemented efficiently using linked-list data structures combined with arrays to store the values of the grid points and their states as shown in Figure 9. This requires only those grid points whose values are changing, the active points and their neighbors, to be visited at each time step. The computation time grows as m^2, where m is the number of grid points along one dimension of U (sometimes called the resolution of the discrete sampling). The m^2 growth in computation time for the sparse-field models is consistent with conventional (parameterized) models, for which computation times increase with surface area rather than volume.

Another advantage of the sparse-field approach is resolution. Equation 17 describes a process whereby all of the level sets of ϕ are pushed toward the zero-set of γ_B. The result is a *shock*, a discontinuity in ϕ. In discrete volumes these shocks take the form of high-contrast areas, which cause aliasing in the resulting models. This results in surface models that are unacceptable for many computer graphics applications, and which do not resemble the target in the final stages of the morph (violating criteria 3 in Section 1).

When using the sparse-field method, the active points serve as a set of control points on the level set. Changing the values of these voxels changes the position of the level set. The forcing function is sampled not at the grid point, but at the location of the nearest level set, which generally lies between grid points. Using a first-order approximation to ϕ produces results that avoid the aliasing problem associated with the shocks

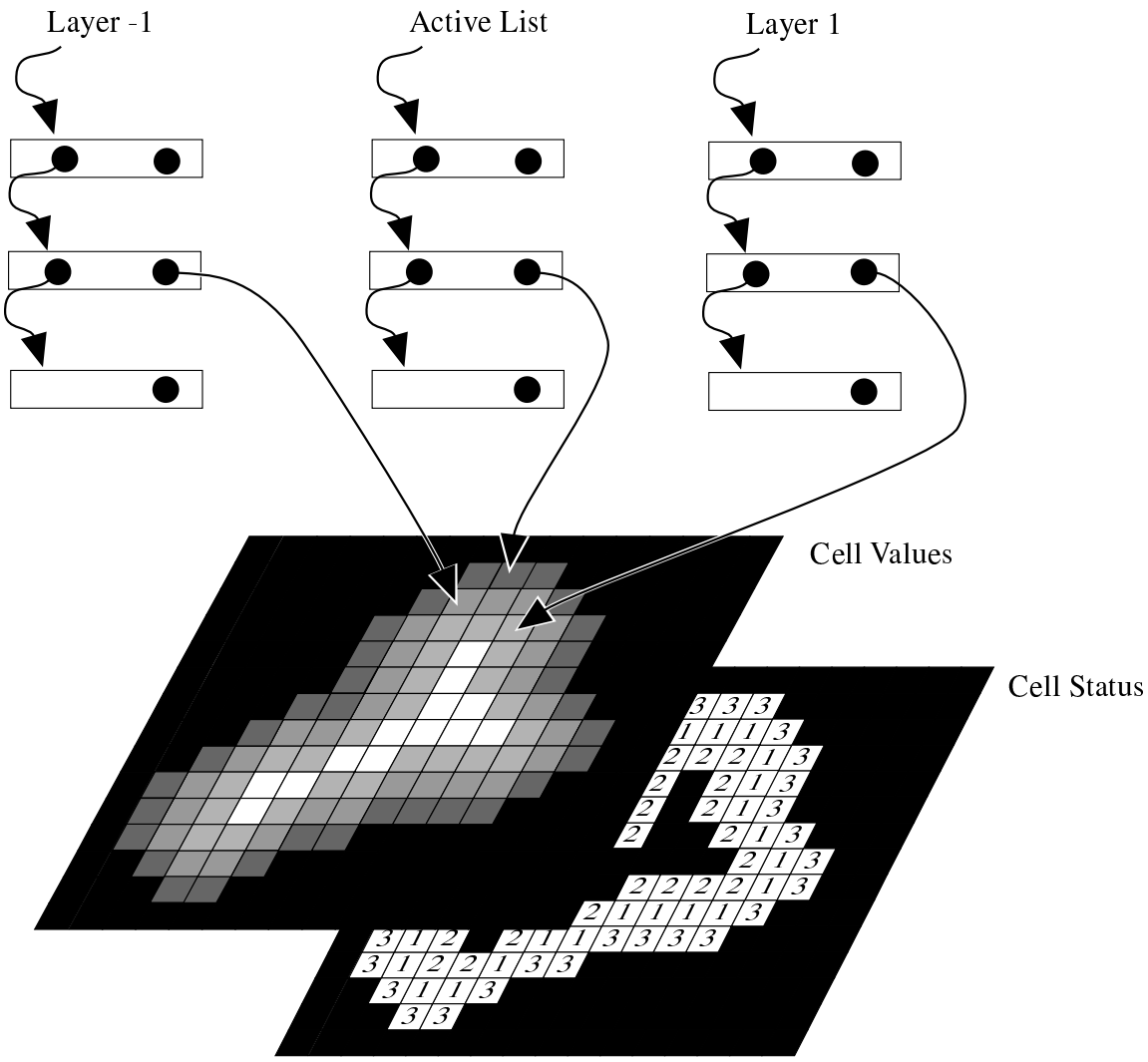


Figure 9: Linked-list data structures provide efficient access to those grid points with values and status that must be updated.

that typically occur with level-set models. Previous work has shown significant increases in the accuracy of fitting level-set models using the first-order modification to the sparse-field method [42], which is essential to the shape metamorphosis application in this paper.

With the first-order modification, the procedure for updating the image and the active set based on surface movements is as follows:

1. For each active grid point (i, j, k):

   (a) Use first-order derivatives and Newton's method to calculate the position (i', j', k') of the nearest zero-crossing to (i, j, k).

   (b) Calculate the $\gamma_B(i', j', k')$ using trilinear interpolation, if necessary.

   (c) Compute the net change of $u^n_{i,j,k}$, based on $\gamma_B(i', j', k')$ and the values of its derivatives using the up-wind scheme (Equation 23).

2. For each active grid point (i, j, k) add the change to the grid point value and determine if the new value $u^{n+1}_{i,j,k}$ falls outside the $[-\frac{1}{2}, \frac{1}{2}]$ interval. If so, put (i, j, k) on lists of grid points that are changing status, called the *status list*; S_1 or S_{-1}, for $u^{n+1}_{i,j,k} > \frac{1}{2}$ or $u^{n+1}_{i,j,k} < -\frac{1}{2}$, respectively.

3. Visit the grid points in the $2N$ layers L_ℓ in the order $\ell = \pm 1, \ldots \pm N$, and update the grid point values based on the values (by adding or subtracting one unit) of the next inner layer, $L_{\ell \mp 1}$. If more than one $L_{\ell \mp 1}$ neighbor exists then use the neighbor that indicates a level set closest to that grid point, i.e., use the maximum for the outside layers and minimum for the inside layers. If a grid point in layer L_ℓ has no $L_{\ell \mp 1}$ neighbors, then it is demoted to $L_{\ell \pm 1}$, the next level away from the active set.

4. For each status list $S_{\pm 1}, S_{\pm 2}, \ldots, S_{\pm N}$:

   (a) For each element (i, j, k) on the status list S_ℓ, remove (i, j, k) from the list $L_{\ell \mp 1}$, and add it to the L_ℓ list, or, in the case of $\ell = \pm(N + 1)$, remove it from all lists.

   (b) Add all $L_{\ell \mp 1}$ neighbors to the $S_{\ell \pm 1}$ list.

More details on sparse-field method and its properties can be found in [40, 42].

6 Summary of the Level-Set Metamorphosis Approach

This section describes the complete approach, based on level-set models, to 3-D shape metamorphosis which meets the criteria listed in the introduction. The specific steps of our level-set morphing approach are 1) 3-D scan conversion of source and target objects, 2) application of coordinate transformations, 3) level-set deformation, and 4) polygonization and rendering.

3-D Scan Conversion. The essential input to the deformation stage of our morphing approach is two 3-D models represented as distance volumes. A distance volume (or distance transform) is a volume dataset where the value stored at each voxel is the shortest distance to the surface of the object being represented by the volume. In our examples, distance volumes are generated by scan converting CSG models, but any technique that converts a solid model into a distance volume may be used. Several methods have been developed for converting polygonal, swept and volumetric models into distance volumes[11, 16, 20, 32, 35, 37].

We have developed a 3-D scan conversion technique that produces a distance volume from a CSG model consisting of superellipsoids [3] and calculates distance to subvoxel accuracy [7, 8]. The distance volume is generated in a two step process. The first step calculates the shortest distance to the CSG model at a set of points within a narrow band around the evaluated surface. Additionally, a second set of points, labeled the zero set, which lies on the CSG model's surface is computed. A point in the zero set is associated with each point in the narrow band. Once the narrow band and zero set are calculated a fast marching method [36, 38] is employed to propagate the shortest distance and closest point information out to the remaining voxels in the volume.

Controlling the Morph with Coordinate Transformations. In order for our active level-set model to deform from one surface into another the source and target objects must overlap. The objects may be automatically or interactively positioned, as well as, interactively warped in order to produce a particular model alignment. The user may choose any of these methods depending upon the level of control and final output desired. The source object will shrink in those areas where it is outside the target object, and will expand in those areas inside the target model. Thus, the user controls the morph by defining the regions of overlap between the source and the target. This is accomplished by applying a coordinate transformation

which maps the voxel locations of the source object into new locations in the target object's distance volume. The transformation is given by

$$\mathbf{x}' = T(\mathbf{x}, \alpha), \tag{25}$$

where $0 \leq \alpha \leq 1$ parameterizes a continuous family of transformations that begins with identity, i.e. $\mathbf{x} = T(\mathbf{x}, 0)$, and smoothly becomes the user-defined transformation at $T(\mathbf{x}, 1)$. The parameterization is utilized during the polygonization stage and is explained further on in this section.

For this work, we have developed a software tool that allows a user to interactively position, rotate and scale the source and target objects in order to produce the transformation T. The coordinate systems of the two objects are aligned, and the user is able to manipulate the objects until they are properly overlapped. We have also developed a technique for automatically positioning, orienting and scaling objects, using 3-D moments, in order to achieve a significant correspondence between two objects without any user input. This method is detailed in the Appendix.

A generalized warping, as defined in [11, 26], may also be employed to provide even more detailed control of the process. Here, the user specifies the numerous geometric features which correspond in the source and target objects. The morph may be predominantly controlled by the geometric warp which has been interactively defined by the user and which has a number of degrees of freedom that are proportional to the number of fiducials. In this case the level-set model would simply fine-tune the surface model as the source object is incrementally transformed by the warping algorithm into the target object. Because the goal of our work is to demonstrate a more powerful blending mechanism that performs well without extensive user input, we do not utilize such generalized warpings for the morphing results presented in this paper.

Level-Set Deformation. Once the overlap of the source and target objects has been defined and any generalized warping has been applied to the source, the level-set deformation process, as described in Sections 3, 4 and 5, is initiated. The process produces a sequence of volume datasets that represent the morphing object. The user defines how often the level-set volume is written to disk during the deformation process.

Polygonization and Rendering. In order to view the morph, we extract a polygonal iso-surface (with the Marching Cubes algorithm [27]) from each volume produced by the level-set deformation process. The

polygons are rendered to produce a series of images, which are then combined to produce an animation. Once the level-set models have been converted into polygons, any number of conventional rendering and animation techniques may be used to shade and view the morphing object. We have developed a color shading method, based on scan-converted closest-point and color information, in order to define the colors on the resulting unparameterized polygonal models. Using this method, the color at any point in space is defined as the color at the closest point on the associated CSG model. We interpolate the color values computed from the source and target models to produce the surface colors for the intermediate shapes. The color shading method is beyond the scope of this paper and is described in [6, 8].

If a shape-changing transformation $T(\mathbf{x}, \alpha)$ (e.g., a scaling, or a generalized warp) has been utilized during the deformation process, the transformation must be interpolated and incrementally applied to the resulting polygonal models generated at each time step. Applying such a transformation implies that the morph is a combination of the user-defined transformation and the level-set deformation. The total time of the morph is scaled down to a range of $[0, 1]$, which matches the parameterization of $T(\mathbf{x}, \alpha)$. For example, a particular morph may produce N time steps, and therefore N individual volume datasets. Before rendering the polygonal model produced from step n, the polygons should be transformed by $T(\mathbf{x}, n/(N-1))$ before being rendered. The number $(N-1)$ results from starting the count at zero. At the final frame, the transformation is $T(\mathbf{x}, 1) = T$, the same transformation that is used to generate γ_B. Thus, the level-set deformation ensures that the source evolves into the target, to within a coordinate transformation, T, which is accounted for by transforming all of the points in the polygonal model.

7 Results

Figure 12 presents a morphing sequence of a dart becoming an X-29 jet. The X-29 and dart models were constructed and scan converted into distance volumes of resolution $96 \times 192 \times 240$ with The Clockworks [15], a CSG modeling system. Lerios et al. [26] demonstrate a similar transition with a jet and a dart, which required the specification of 37 different user-defined correspondence elements on both models, roughly 200 user-defined parameters. Our morph required only a few minutes of user time to interactively overlay the source and target models. The jet and dart have been rendered semi-transparently in their initial configurations and presented in Figure 10 in order to demonstrate how they were overlaid before initiating

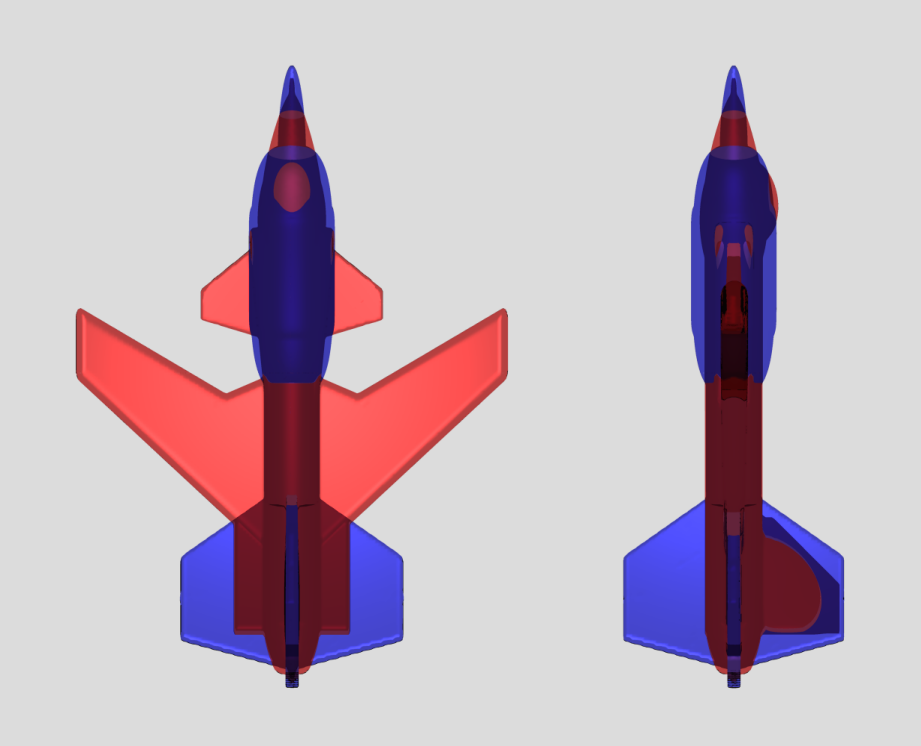


Figure 10: Initial model configuration for morph in Figures 12 and 13.

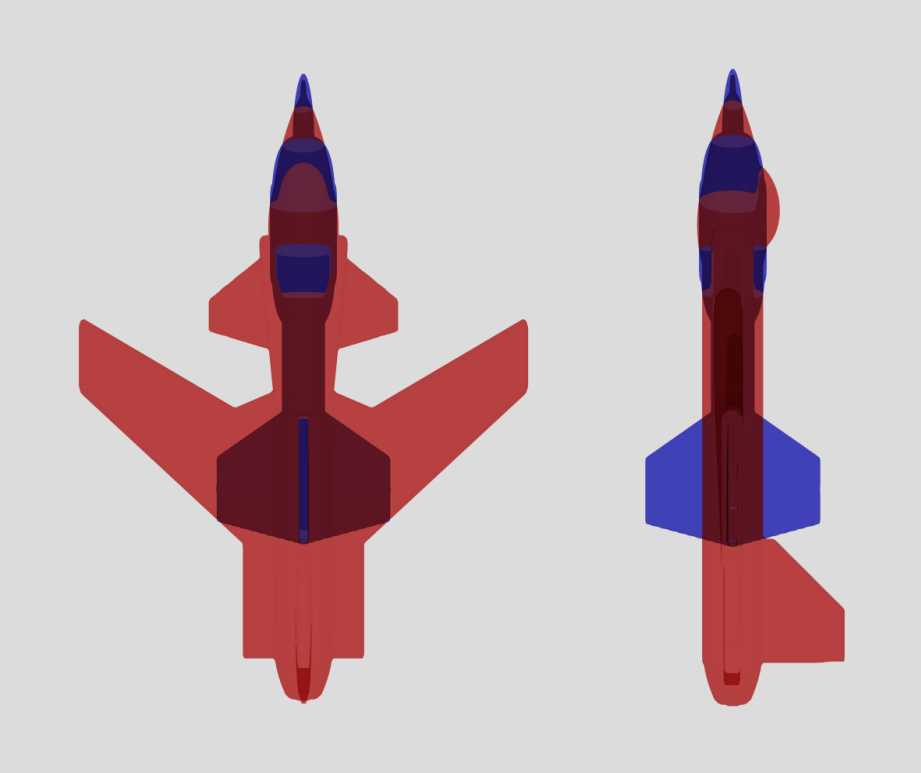


Figure 11: Initial model configuration for morph in Figure 14.

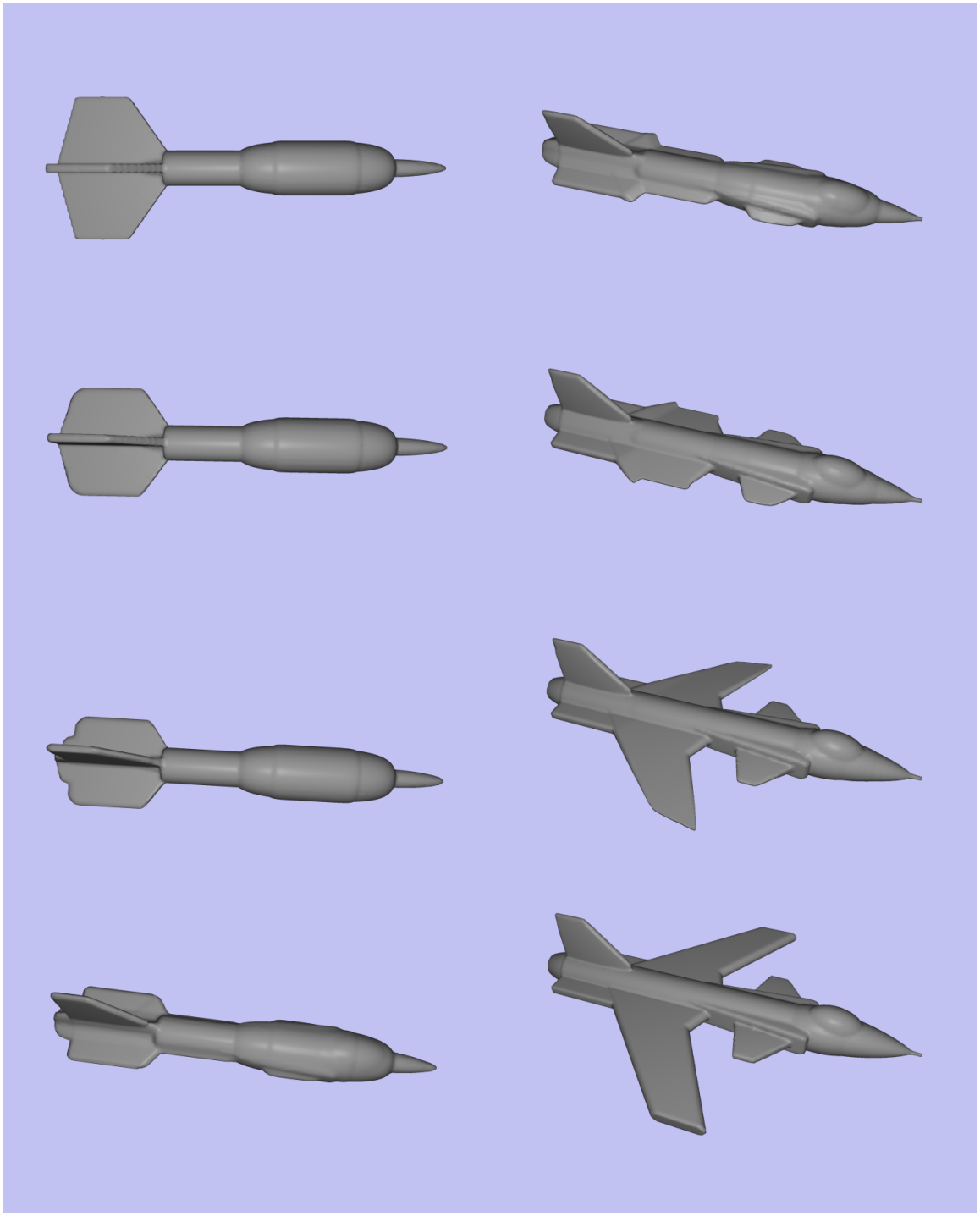


Figure 12: A dart morphing into an X-29 jet.

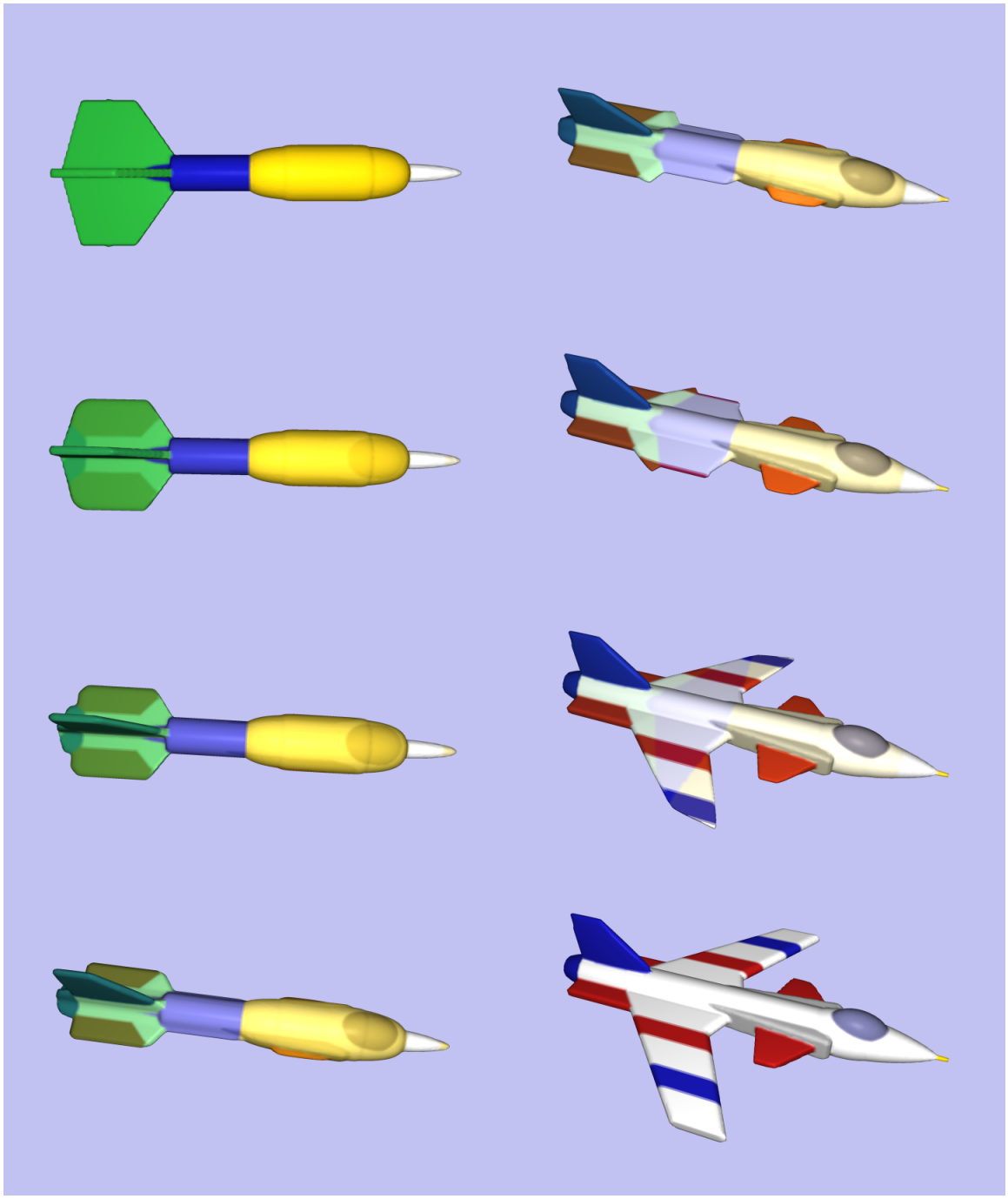


Figure 13: A dart morphing into an X-29 jet with interpolating surface colors.

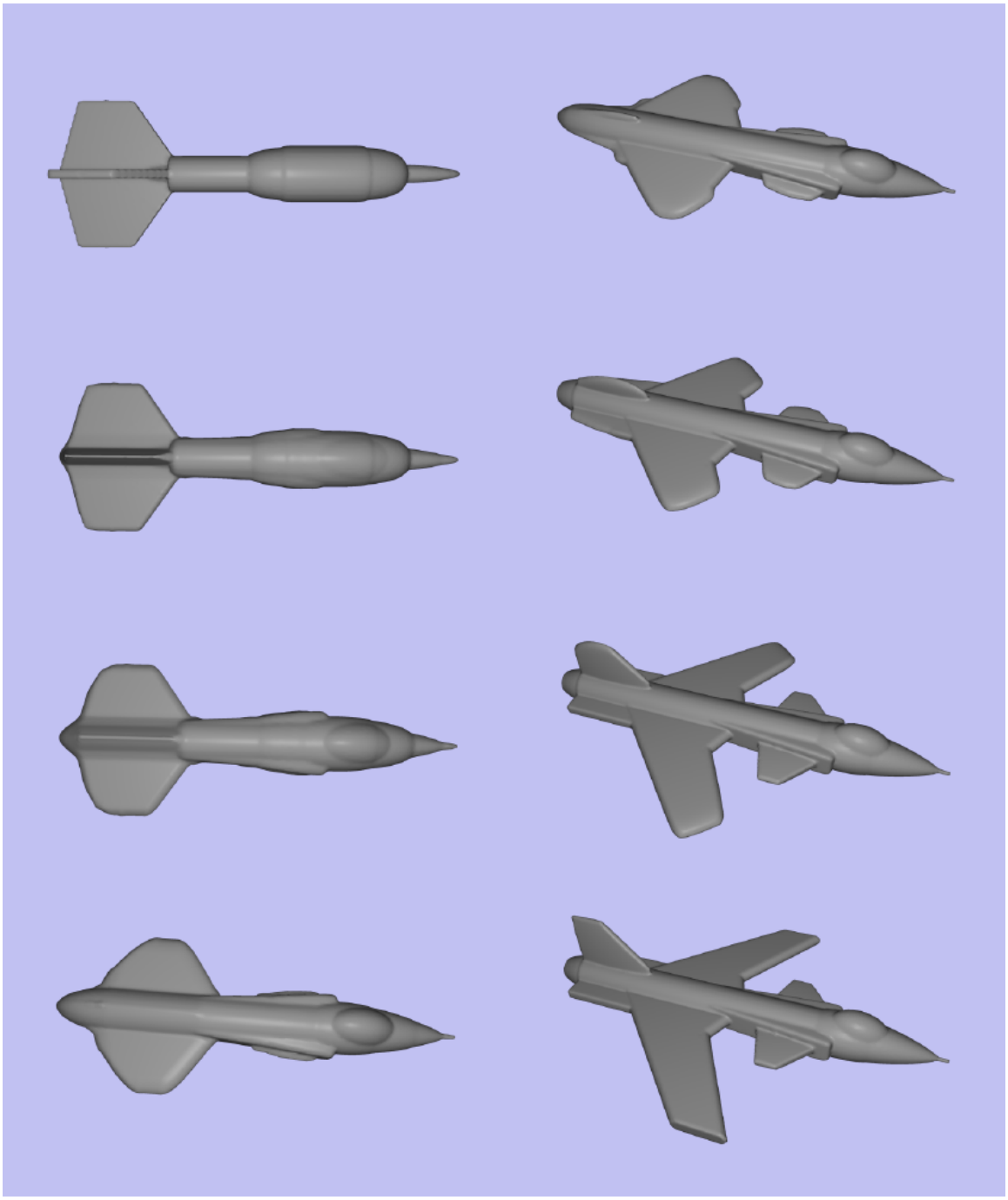


Figure 14: A dart morphing into an X-29 jet using different initial conditions.

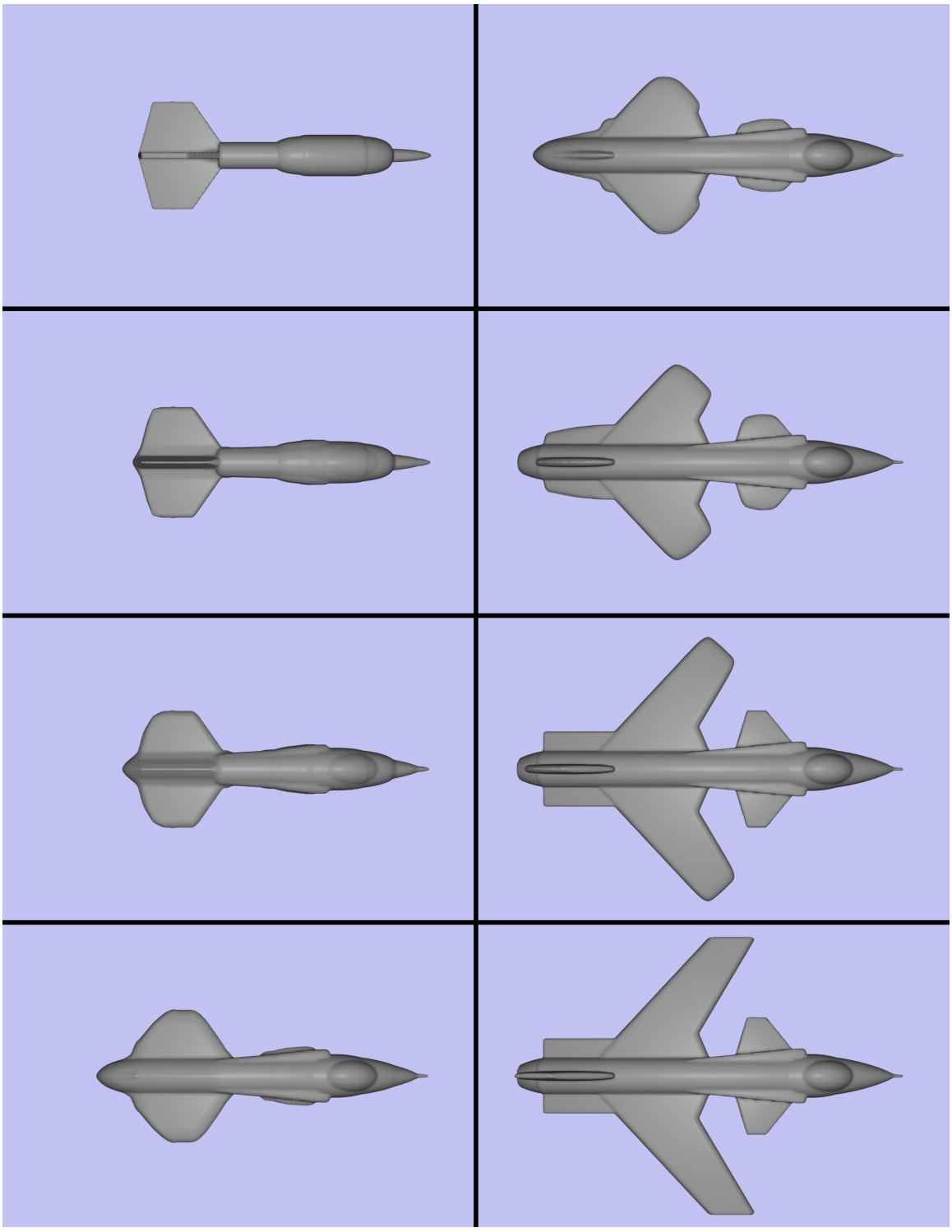


Figure 15: The morph from Figure 14 without applying the incremental transformation.

Figure 16: Initial model configuration for morph in Figure 17.

the deformation process. The jet was rendered in light transparent red, and the dart was rendered in light transparent blue. The areas of dark red or dark blue indicate regions where the models overlap. Figure 13 presents the morphing sequence of Figure 12 with interpolating surface colors generated with our color-shading algorithm [6, 8].

Figure 14 uses the same input models with slightly different initial conditions to produce a different morph of the dart turning into the X-29. In this morph the animator wanted the back fins of the dart to morph into the wings of the jet. This was achieved with only a few minutes of user input, the time needed to specify a scale and translation applied to the dart. The level-set deformation stage for this (and the previous) morphing sequence required approximately 9 CPU-minutes on an SGI R10000 Onyx2. Figure 11 presents the initial conditions of the jet and dart model for the second morph. The dart has been scaled by .75 and translated by $[11.9, 24.0, -7.2]$ so that its fins will overlap with the wings of the jet. Before rendering each frame of the morphing sequence, each vertex ($\mathbf{P}$) of the polygonal model produced by the Marching Cubes algorithm is transformed by

$$\mathbf{P}' = (1.0 - ((n/(N-1))(1.0 - 0.75)))\mathbf{P} + (n/(N-1))[11.9, 24.0, -7.2], \tag{26}$$

where n is the frame number and N is the total number of frames. An additional global rotation has been applied to the models in Figures 12, 13, 14 and 17 to highlight the three-dimensional structure of the

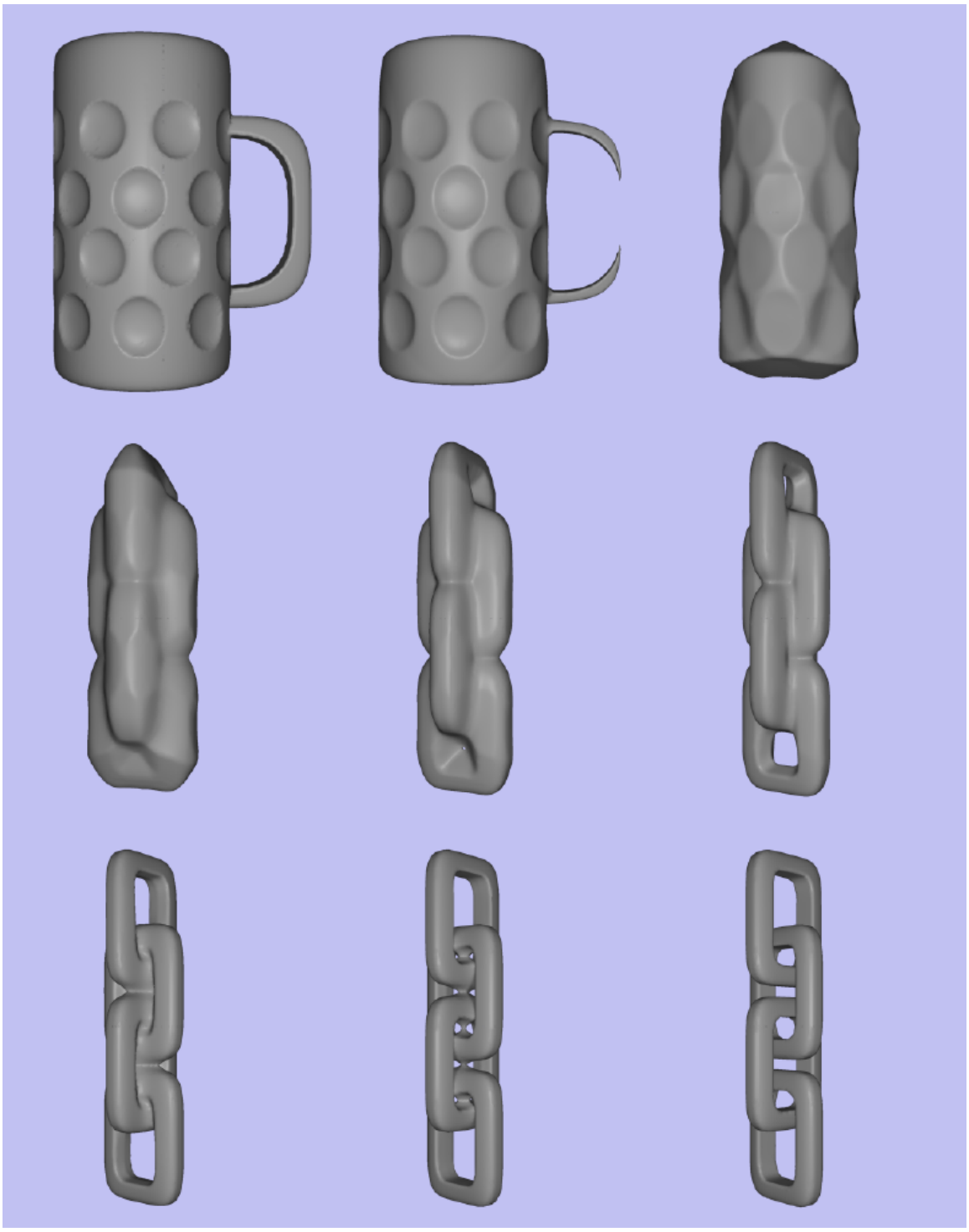


Figure 17: A mug morphing into a chain. This demonstrates that level set models easily cope with changes in topology.

morphing model. Applying the transformation in Equation 26 ensures that the size and location of the morphing model remains approximately constant while it is changing shape. Figure 15 presents the morph from Figure 14 without applying the transformation. It can be seen that without the transformation the morphing model changes shape *and* size, with the tail end of the dart growing into the rear section of the X-29. As seen in Figure 10 the dart and X-29 initially are approximately the same size.

Figure 16 presents the initial configuration of a mug-to-chain morph. The mug and chain were originally defined with CSG models and then scan converted into distance volumes of resolution $120 \times 148 \times 184$. Figure 17 presents the morphing result. The level-set deformation stage of this sequence required approximately 20 CPU-minutes on an SGI R10000 Onyx2. This sequence demonstrates that the level-sets approach easily copes with changes of topology during morphing. The results from all three morphing sequences also highlight the advantage of calculating to sub-voxel accuracy. The volume resolutions are somewhat low, but the deforming surfaces extracted from them are mostly free of aliasing artifacts.

8 Conclusions and Future Work

This paper has presented a volume-based method for achieving 3-D shape metamorphosis. The method relies on a novel approach to the blending stage of the morphing process. This stage is formulated as the optimization, via a hill-climbing strategy, of a similarity measure between the deforming surface and the target, utilizing level-set models for the incremental shape changes. These models take advantage of a volume-based representation to calculate surface deformations to sub-voxel accuracy. The movements of these deformable models are driven by the signed distance transform of the target, which is also computed to sub-voxel accuracy. The result is a 3-D morphing technique which demonstrates outstanding fidelity, level of detail, flexibility, and degree of automation, comparing favorably with other methods in the literature.

As with all volumetric morphing methods, our method has its limitations. The basic volumetric representation of the objects can produce aliasing artifacts on objects that have regions of high curvature. Since a distance volume is a sampled representation, the accuracy of individual object features is restricted by the sampling resolution of volume. Additionally, our method is only useful for solid, (i.e. closed) objects. Our method cannot be used to morph open shell-like surfaces.

Even given our current results, more work remains. We will continue to work on our volume-based technique for including texture maps or surface coloring in the 3-D morph. We have also developed methods for creating distance transforms from 3-D polygonal meshes and MR/CAT-generated volume datasets. We will use these capabilities to generate morphing sequences between different types of models. Future work will also focus on better computational schemes, including parallel processing, in order to achieve 3-D morphs at interactive rates.

9 Acknowledgements

We would like to thank Dr. Alan Barr and the other members of the Caltech Computer Graphics Group for their support and assistance. Sean Mauch developed the Fast Marching algorithm and software that was used for scan converting the models in the morphing examples. Timothy Doyle created the dart model. This work was financially supported by the National Science Foundation (ASC-89-20219), as part of the STC for Computer Graphics and Scientific Visualization; the National Institute on Drug Abuse, the National Institute of Mental Health and the NSF, as part of the Human Brain Project; and the Volume Visualization Program of the Office of Naval Research (N00014-97-0227). Additional equipment grants were provided by SGI, Hewlett-Packard, IBM, and Digital Equipment Corporation. This work was initially funded by the former shareholders of the European Computer-Industry Research Centre: Bull SA, ICL PLC, and Siemens AG.

References

[1] David Adalstein and James A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, pages 269–277, 1995.

[2] T.F. Banchoff and P.J. Giblin. Global theorems for symmetry sets of smooth curves and polygons in the plane. *Proc. Royal Soc. Edinburgh*, 106A:221–231, 1987.

[3] A. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981.

[4] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 35–42, July 1992.

[5] Harry Blum and Roger N. Nagel. Shape description using weighted symmetric axis features. *Pattern Recognition*, 10:167–180, 1978.

[6] D. Breen and S. Mauch. Generating shaded offset surfaces with distance, closest-point and color volumes. In *Proceedings of the International Workshop on Volume Graphics*, pages 307–320, March 1999.

[7] D. Breen, S. Mauch, and R. Whitaker. 3D scan conversion of CSG models into distance volumes. In *Proceedings of the 1998 Symposium on Volume Visualization*, pages 7–14. ACM SIGGRAPH, October 1998.

[8] David Breen, Sean Mauch, and Ross Whitaker. 3D scan conversion of CSG models into distance, closest-point and color volumes. In M. Chen, A. Kaufman, and R. Yagel, editors, *Volume Graphics*, chapter 8. Springer, London, 2000.

[9] David T. Chen, Andrei State, and David Banks. Interactive shape metamorphosis. In P. Hanrahan and J. Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 43–44. ACM SIGGRAPH, April 1995.

[10] M. Chen, M.W. Jones, and P. Townshend. Volume distortion and morphing using disk fields. *Computers & Graphics*, 20(4):567–575, 1996.

[11] D. Cohen-Or, D. Levin, and A. Solomivici. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, 1998.

[12] D. DeCarlo and J. Gallier. Topological evolution of surfaces. In *Proceedings of Graphics Interface '96*, pages 194–203, May 1996.

[13] C. Fox. *Introduction to Calculus of Variations*. Dover Publications, New York, 1987.

[14] E. Galin and S. Akkouche. Blob metamorphosis based on minkowski sums. *Computer Graphics Forum*, 15(3):143–153, 1996. Eurographics '96 Conference issue.

[15] P. Getto and D. Breen. An object-oriented architecture for a computer animation system. *The Visual Computer*, 6(2):79–92, March 1990.

[16] S. Gibson. Using distance maps for accurate surface representation in sampled volumes. In *Proceedings of the 1998 Symposium on Volume Visualization*, pages 23–30. ACM SIGGRAPH, October 1998.

[17] A. Gregory, A. State, M. Lin, D. Manocha, and M. Livingston. Feature-based surface decomposition for correspondence and morphing between polyhedra. In *Proceedings of Computer Animation*, pages 64–71, June 1998.

[18] T. He, S. Wang, and A. Kaufman. Wavelet-based volume morphing. In *Proceedings of IEEE Visualization '94*, pages 85–92. IEEE Press, October 1994.

[19] John F. Hughes. Scheduled Fourier volume morphing. In E.E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 43–46, July 1992.

[20] M.W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, 1996.

[21] Takashi Kanai, Hiromasa Suzuki, and Fumihiko Kimura. Three-dimensional geometric metamorphosis based on harmonic maps. *The Visual Computer*, 14(4):166–176, 1998.

[22] Anil Kaul and Jarek Rossignac. Solid-interpolating deformations: Construction and animation of PIPs. *Computers & Graphics*, 16(1):107–116, 1992.

[23] James R. Kent, Wayne E. Carlson, and Richard E. Parent. Shape transformation for polyhedral objects. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 47–54, July 1992.

[24] Francis Lazarus and Anne Verroust. Metamorphosis of cylinder-like objects. *Journal of Visualization and Computer Animation*, 8:131–146, 1997.

[25] Aaron W.F. Lee, David Dobkin, Wim Sweldens, and Peter Schröder. Multiresolution mesh morphing. In *SIGGRAPH '99 Conference Proceedings*, Annual Conference Series, pages 343–350. Addison Wesley, August 1999.

[26] Apostolos Lerios, Chase D. Garfinkle, and Marc Levoy. Feature-Based volume metamorphosis. In Robert Cook, editor, *SIGGRAPH '95 Conference Proceedings*, Annual Conference Series, pages 449–456. Addison Wesley, August 1995.

[27] W.E. Lorensen and H.E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In M.C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[28] Ravikanth Malladi, James A. Sethian, and Baba C. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):158–175, 1995.

[29] James V. Miller, David E. Breen, William E. Lorensen, Robert M. O'Bara, and Michael J. Wozny. Geometrically deformed models: A method for extracting closed geometric models from volume data. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 217–226, July 1991.

[30] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[31] Richard Parent. Shape transformation by boundary representation interpolation: A recursive approach to establishing face correspondences. *Journal of Visualization and Computer Animation*, 3:219–239, 1992.

[32] B. Payne and A. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, 12(1):65–71, 1992.

[33] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes in C (2nd edition)*. Cambridge University Press, New York, NY, 1992.

[34] J. R. Rossignac and A. Kaul. AGRELs and BIPs: Metamorphosis as a bezier curve in the space of polyhedra. *Computer Graphics Forum*, 13(3):179–184, 1994. Eurographics '94 Conference issue.

[35] W. Schroeder, W. Lorensen, and S. Linthicum. Implicit modeling of swept surfaces and volumes. In *Proceedings of IEEE Visualization '94*, pages 40–45. IEEE Press, October 1994.

[36] J.A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Science*, volume 93 of 4, pages 1591–1595, 1996.

[37] J.A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, UK, 1999.

[38] J.N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[39] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *SIGGRAPH '99 Conference Proceedings*, Annual Conference Series, pages 335–342. Addison Wesley, August 1999.

[40] R. Whitaker. Algorithms for implicit deformable models. In *Proceedings of the Fifth International Conference on Computer Vision*, pages 822–827. IEEE, June 1995.

[41] R. Whitaker and D. Breen. Level-set models for the deformation of solid objects. In *Proceedings of the Third International Workshop on Implicit Surfaces*, pages 19–35. Eurographics Association, June 1998.

[42] Ross T. Whitaker. A level-set approach to 3D reconstruction from range data. *International Journal of Computer Vision*, 29(3):203–231, October 1998.

Appendix Automatic Object Alignment

The user may allow the system to automatically calculate the transformation needed to align the source and target objects. While this won't guarantee that the objects will overlap and produce a reasonable morph, it does provide a way for a user to easily create an initial configuration for the morphing sequence. The alignment is accomplished by calculating two affine transformations (consisting of rotation, translation and scaling) from gross geometric measures, which map a point in the global coordinate system of each of the objects into each of their intrinsic coordinate systems. We use the moments of the objects, calculated on point samples, to construct the transformation for each object. The centroid and principal axes of the objects define local coordinate systems for those objects, which we assume are aligned with each other.

The centroid, $\bar{\mathbf{p}}$, of an object is the average position of its internal points. The object is sampled on a regular grid within its bounding box. Each grid point is tested to determine if it is inside or outside the object. If $\mathcal{V}$ is the set of coordinates of internal points and n is the number of points in that set, then

$$\bar{\mathbf{p}} = \frac{1}{n} \sum_{\mathbf{p} \in \mathcal{V}} \mathbf{p}. \tag{27}$$

The principal axes of the objects are the eigenvectors of the covariance matrix associated with each object. The covariance matrix is defined as

$$\mathbf{C} = \begin{pmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{pmatrix}, \tag{28}$$

where

$$c_{ij} = \frac{1}{n} \sum_{\mathbf{p}_i, \mathbf{p}_j \in \mathcal{V}} (\mathbf{p}_i - \bar{\mathbf{p}}_i) * (\mathbf{p}_j - \bar{\mathbf{p}}_j), \tag{29}$$

and $i, j \in \{x, y, z\}$. The eigenvectors $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$, which are orthogonal, and eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$, which

are positive and real, of the covariance matrix $\mathbf{C}$ are calculated with a QL algorithm utilizing implicit shifts [33].

The eigenvalues provide some information about the relative size of the objects along each local axis. The square root of the eigenvalues gives a rough measure of the object's dimensions (exact dimensions if the object is an ellipsoid). The normalized eigenvectors from the two objects are ordered and matched up according to the value of their corresponding eigenvalues. We assume that the principal axis with the greatest eigenvalue is the Z-axis, the second greatest is the Y-axis, and the principal axis with the smallest eigenvalue is the X-axis. The ordered eigenvectors may be "lined up" to construct a rotation matrix which maps a vector in the global coordinate system of the object into the local coordinate system defined by the object's principal axes. Assuming that the eigenvalues $\lambda_1, \lambda_2, \lambda_3$ are ordered in increasing magnitude, the associated rotation matrix is defined as

$$\mathbf{R} = \begin{pmatrix} & & \\ \mathbf{e}_1{}^T & \mathbf{e}_2{}^T & \mathbf{e}_3{}^T \\ & & \end{pmatrix}. \tag{30}$$

The inverse rotation, which maps a vector in an object's local intrinsic coordinate system back into the global coordinate system is simply the transpose of Equation 30,

$$\mathbf{R}^{-1} = \begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \end{pmatrix}. \tag{31}$$

The scaling matrix $\mathbf{S}$ is used to scale the source object Ω_A into the approximate size of the target object Ω_B. It is defined as

$$\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{pmatrix}, \tag{32}$$

where

$$\begin{aligned} S_x &= \sqrt{\lambda_1^B / \lambda_1^A}, \\ S_y &= \sqrt{\lambda_2^B / \lambda_2^A}, \\ S_z &= \sqrt{\lambda_3^B / \lambda_3^A}. \end{aligned} \tag{33}$$

λ_1^A, λ_2^A, λ_3^A, λ_1^B, λ_2^B, λ_3^B are the eigenvalues associated with the eigenvectors $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ of object Ω_A and

object Ω_B.

A point $\mathbf{p}$ in the global coordinate system of object Ω_A may be mapped into the local intrinsic coordinate system of object Ω_B ($\mathbf{p}'$) imbedded in B's global coordinate system by first subtracting the centroid of A, $\bar{\mathbf{p}}^A$, then applying the rotation matrix $\mathbf{R^A}$. This places point $\mathbf{p}$ into the intrinsic local coordinate system of A, which is assumed to be aligned with B's. The scaling matrix $\mathbf{S}$ is then applied so that the general dimensions of object Ω_A are approximately the same as object Ω_B's. The rotation matrix $\mathbf{R^{B^{-1}}}$ is applied, and the point is shifted by $\bar{\mathbf{p}}^B$, the centroid of B, which maps the point into the global coordinate system of B. The transformation steps may be summarized as

$$\mathbf{p}' = (\mathbf{p} - \bar{\mathbf{p}}^A)\mathbf{R^A}\mathbf{S}(\mathbf{R^B})^{-1} + \bar{\mathbf{p}}^B. \tag{34}$$

Level Set Surface Editing Operators

Ken Museth* David E. Breen* Ross T. Whitaker† Alan H. Barr*

*Computer Science Department
California Institute of Technology

†School of Computing
University of Utah

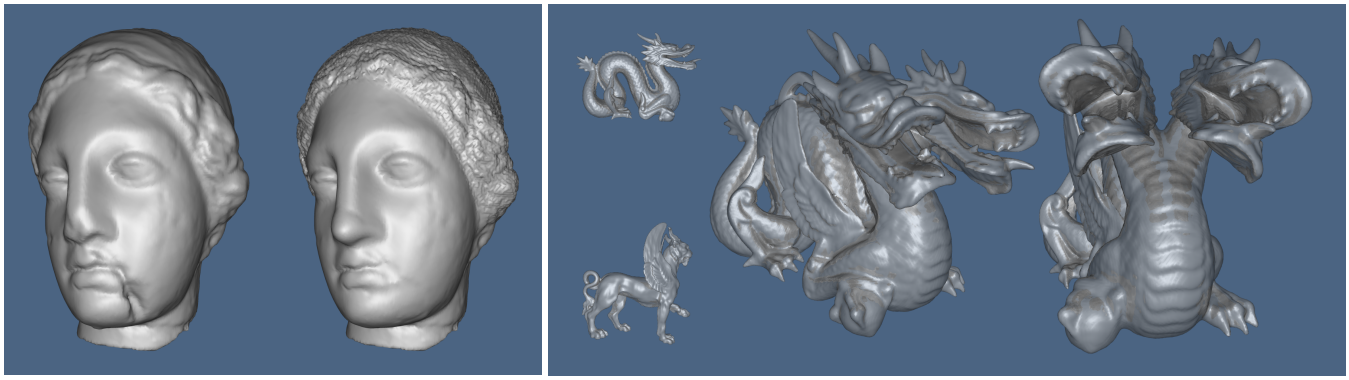


Figure 1: Surfaces edited with level set operators. Left: A damaged Greek bust model is repaired with a new nose, chin and sharpened hair. Right: A new model is constructed from models of a griffin and dragon (small figures), producing a two-headed, winged dragon.

Abstract

We present a level set framework for implementing editing operators for surfaces. Level set models are deformable implicit surfaces where the deformation of the surface is controlled by a speed function in the level set partial differential equation. In this paper we define a collection of speed functions that produce a set of surface editing operators. The speed functions describe the velocity at each point on the evolving surface in the direction of the surface normal. All of the information needed to deform a surface is encapsulated in the speed function, providing a simple, unified computational framework. The user combines pre-defined building blocks to create the desired speed function. The surface editing operators are quickly computed and may be applied both regionally and globally. The level set framework offers several advantages. 1) By construction, self-intersection cannot occur, which guarantees the generation of physically-realizable, simple, closed surfaces. 2) Level set models easily change topological genus, and 3) are free of the edge connectivity and mesh quality problems associated with mesh models. We present five examples of surface editing operators: blending, smoothing, sharpening, openings/closings and embossing. We demonstrate their effectiveness on several scanned objects and scan-converted models.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Surface and object representations; I.3.4 [Computer Graphics]: Graphics Utilities—Graphics Editors;

Keywords: Deformations, geometric modeling, implicit surfaces, shape blending.

1 Introduction

The creation of complex models for such applications as movie special effects, graphic arts, and computer-aided design can be a time-consuming, tedious, and error-prone process. One of the solutions to the model creation problem is 3D photography [Bouguet and Perona 1999], i.e. scanning a 3D object directly into a digital representation. However, the scanned model is rarely in a final desired form. The scanning process is imperfect and introduces errors and artifacts, or the object itself may be flawed.

3D scans can be converted to polygonal and parametric surface meshes [Edelsbrunner and Mücke 1994; Bajaj et al. 1995; Amenta et al. 1998]. Many algorithms and systems for editing these polygonal and parametric surfaces have been developed [Cohen et al. 2001], but surface mesh editing has its limitations and must address several difficult issues. For example, it is difficult to guarantee that a mesh model will not self-intersect when performing a local editing operation based on the movement of vertices or control points, producing non-physical, invalid results. See Figure 2. If self-intersection occurs, it must be fixed as a post-process. Also, when merging two mesh models the process of clipping individual polygons and patches may produce errors when the elements are small and/or thin, or if the elements are almost parallel. In addition while it is not impossible to change the genus of a surface mesh model [Biermann et al. 2001], it is certainly difficult and requires significant effort to maintain the consistency/validity of the underlying vertex/edge connectivity structure.

1.1 New Surface Editing Operators

In order to overcome these difficulties we present a level set approach to implementing operators for locally and globally editing closed surfaces. Level set models are deformable implicit surfaces that have a volumetric representation [Osher and Sethian 1988]. They are defined as an iso-surface, i.e. a level set, of some implicit function ϕ. The surface is deformed by solving a partial differential equation (PDE) on a regular sampling of ϕ, i.e. a volume dataset. To date level set methods have not been developed for adaptive grids, a limitation of current implementations, but not of the mathematics. It should be emphasized that level set methods do not manipulate an explicit closed form representation of ϕ, but only

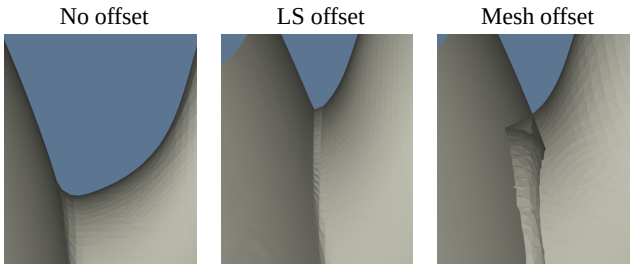


Figure 2: (left) A cross-section of the teapot model near the spout. (middle) No self-intersection occurs, by construction, when performing a level set (LS) offset, *i.e.* dilation, of the surface. (right) Self-intersections may occur when offsetting a mesh model.

a sampling of it. Level set methods provide the techniques needed to change the voxel values of the volume in a way that deforms the embedded iso-surface to meet a user-defined goal. The user controls the deformation of the level set surface by defining a speed function $\mathcal{F}(\boldsymbol{x}, \ldots)$, the speed of the level set at point $\boldsymbol{x}$ in the direction of the normal to the surface at $\boldsymbol{x}$. Therefore all the information needed to deform a level set model may be encapsulated in a single speed function $\mathcal{F}()$, providing a simple, unified computational framework.

We have developed a number of surface editing operators within the level set framework by defining a collection of new level set speed functions. The cut-and-paste operator (Section 5.1) gives the user the ability to copy, remove and merge level set models (using volumetric CSG operations) and automatically blends the intersection regions (See Section 5.2). Our smoothing operator allows a user to define a region of interest and smooths the enclosed surface to a user-defined curvature value. See Section 5.3. We have also developed a point-attraction operator. See Section 5.4. Here, a regionally constrained portion of a level set surface is attracted to a single point. By defining line segments, curves, polygons, patches and 3D objects as densely sampled point sets, the single point attraction operator may be combined to produce a more general surface embossing operator. As noted by others, the opening and closing morphological operators may be implemented in a level set framework [Sapiro et al. 1993; Maragos 1996]. We have also found them useful for performing global blending (closing) and smoothing (opening) on level set models. Since all of the operators accept and produce the same volumetric representation of closed surfaces, the operators may be applied repeatedly to produce a series of surface editing operations. See Figure 11.

1.2 Benefits and Issues

Performing surface editing operations within a level set framework provides several advantages and benefits. Many types of surfaces may be imported into the framework as a distance volume, a volume dataset that stores the signed shortest distance to the surface at each voxel. This allows a number of different types of surfaces to be modified with a single, powerful procedure. By construction, the framework always produces non-self-intersecting surfaces that represent physically-realizable objects, an important issue in computer-aided design. Level set models easily change topological genus, and are free of the edge connectivity and mesh quality problems associated with deforming and modifying mesh models. Additionally, some reconstruction algorithms produce volumetric models [Curless and Levoy 1996; Whitaker 1998; Zhao et al. 2001] and volumetric scanning systems are increasingly being employed in a number of diverse fields. Therefore volumetric models are becoming more prevalent and there is a need to develop powerful editing operators that act on these types of models directly.

There are implementation issues to be addressed when using level set models. Given their volumetric representation, one may be concerned about the amount of computation time and memory needed to process level set models. Techniques have been developed to limit level set computations to only a narrow band around the level set of interest [Adalsteinsson and Sethian 1995; Whitaker 1998; Peng et al. 1999] making the computational complexity proportional to the surface area of the model. We have also developed computational techniques that allow us to perform the narrow band calculations only in a portion of the volume where the level set is actually moving. Additionally, fast marching methods have been developed to rapidly evaluate the level set equation under certain circumstances [Tsitsiklis 1995; Sethian 1996]. Memory usage has not been an issue when generating the results in this paper. The memory needed for our results (512 MB) is available on standard workstations and PCs. We have implemented our operators in an interactive environment that allows us to easily edit a number of complex surfaces. Additionally, concerns have been raised that volume-based models cannot represent fine or sharp features. Recent advances [Frisken et al. 2000; Kobbelt et al. 2001] have shown that is is possible to model these kinds of structures with volume datasets, without excessively sampling the whole volume. These advances will also be available for our operators once adaptive level set methods, an active research area, are developed.

1.3 Contributions

The major contributions of our work are the following.

- The introduction of a unified approach to surface editing within a level set framework.
  - Editing operators defined by speed functions.
  - Results produced by solving a PDE.
- The definition of level set speed functions that implement blending, smoothing and embossing surface editing operators.
  - Blending is automatic and is constrained to only occur within a user-specified distance to an arbitrarily complex intersection curve.
  - Smoothing and embossing are constrained to occur within a user-specified region.
  - The user specifies the local geometric properties of the resulting surface modifications.
  - The user specifies if material should be added and/or removed during editing operations.
- The new techniques used to localize level set calculations.
- In Appendix B we present a new, numerically-stable curvature measure for level set surfaces.

2 Previous Work

Three areas of research are closely related to our level set surface editing work; volumetric sculpting, mesh-based surface editing/fairing and implicit modeling. Volumetric sculpting provides methods for directly manipulating the voxels of a volumetric model. CSG Boolean operations [Hoffmann 1989; Wang and Kaufman 1994] are commonly found in volume sculpting systems, providing a straightforward way to create complex solid objects by combining simpler primitives. One of the first volume sculpting systems is presented in [Galyean and Hughes 1991]. [Wang and Kaufman 1995] improved on this work by introducing tools for carving and sawing. More recently [Perry and Frisken 2001] implemented a volumetric sculpting system based on Adaptive Distance Fields (ADF) [Frisken et al. 2000], allowing for volumetric models with adaptive resolution.

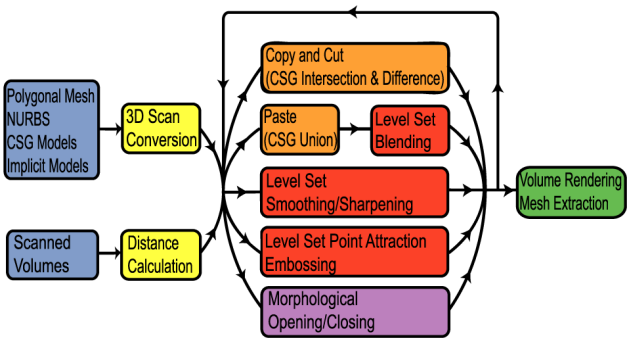


Figure 3: Our level set surface editing operators (red) fit into a larger editing framework. The pipeline consists of: input models (blue), pre-processing (yellow), CSG operations (orange), local LS operators (red), global LS operators (purple) and rendering (green).

Performing CSG operations on mesh models is a long-standing area of research [Requicha and Voelcker 1985; Laidlaw et al. 1986]. Recently CSG operations were developed for multi-resolution subdivision surfaces by [Biermann et al. 2001], but this work did not address the problem of blending or smoothing the sharp features often produced by the operations. However, the smoothing of meshes has been studied on several occasions [Welch and Witkin 1994; Taubin 1995; Kobbelt et al. 1998]. [Desbrun et al. 1999] have developed a method for fairing irregular meshes using diffusion and curvature flow, demonstrating that mean-curvature based flow produces the best results for smoothing.

There exists a large body of surface editing work based on implicit models [Bloomenthal et al. 1997]. This approach uses implicit surface representations of analytic primitives or skeletal offsets. The implicit modeling work most closely related to ours is found in [Wyvill et al. 1999]. They describe techniques for performing blending, warping and boolean operations on skeletal implicit surfaces. [Desbrun and Gascuel 1995] address the converse problem of preventing unwanted blending between implicit primitives, as well as maintaining a constant volume during deformation.

Level set methods have been successfully applied in computer graphics, computer vision and visualization [Sethian 1999; Sapiro 2001], for example medical image segmentation [Malladi et al. 1995; Whitaker et al. 2001], shape morphing [Breen and Whitaker 2001], 3D reconstruction [Whitaker 1998; Zhao et al. 2001], and recently for the animation of liquids [Foster and Fedkiw 2001].

Our work stands apart from previous work in several ways. We have not developed volumetric modeling tools. Our editing operators act on surfaces that happen to have an underlying volumetric representation, but are based on the mathematics of deforming implicit surfaces. Our editing operators share several of the capabilities of mesh-based tools, but are not hampered by the difficulties of maintaining vertex/edge information. Since level set models are not tied to any specific implicit basis functions, they easily represent complex models to within the resolution of the sampling. Our work is the first to utilize level set methods to perform user-controlled editing of complex geometric models.

3 Overview of the Editing Pipeline

The level set surface editing operators should be viewed as components of a larger modeling framework. The pipeline for this framework is presented in Figure 3. The red components contain the level set speed functions that we have developed for localized surface editing. The remaining components contain the data and operations needed for level set modeling, input models (blue), preprocessing (yellow), CSG operations (orange), global LS operators (purple) and rendering (green). The pipeline provides the context for the details of our speed functions.

3.1 Input and Output Models

We are able to import a wide variety of closed geometric models into the level set environment. We represent a level set model as an iso-surface embedded in a distance volume. Frequently we only store distance information in a narrow band of voxels surrounding the level set surface. As illustrated in Figure 3 we have developed and collected a suite of scan conversion methods for converting polygonal meshes, CSG models [Breen et al. 2000], implicit primitives, and NURBS surfaces into distance volumes. Additionally many types of scanning processes produce volumetric models directly, e.g. MRI, CT and laser range scan reconstruction. These models may be brought into our level set environment as is or with minimal pre-processing. We frequently segment these models with another level set technique [Whitaker et al. 2001], and then apply Sethian's Fast Marching Method [Sethian 1996] to convert the results into distance volumes. The models utilized in this paper and their original form are listed in Table 1.

Table 1: Native representations of the input models and dimensions of the corresponding scan converted distance volumes.

Model	Representation	Dimensions
Dragon	volumetric reconstruction	$356 \times 161 \times 251$
Griffin	volumetric reconstruction	$312 \times 148 \times 294$
Greek bust	polygonal reconstruction	$221 \times 221 \times 161$
Human head	polygonal reconstruction	$256 \times 246 \times 193$
Utah teapot	NURBS surface	$156 \times 232 \times 124$
Supertoroid	implicit primitive	$91 \times 91 \times 31$

In the final stage of the pipeline we can either volume render the surface directly or render a polygonal mesh extracted from the volume. While there are numerous techniques available for both approaches, we found extracting and rendering Marching Cubes meshes [Lorensen and Cline 1987] to be satisfactory.

4 Level Set Surface Modeling

The Level Set Method, first presented in [Osher and Sethian 1988], is a mathematical tool for modeling surface deformations. A deformable (*i.e.* time-dependent) surface is implicitly represented as an iso-surface of a time-varying scalar function, $\phi(\boldsymbol{x}, t)$. A detailed description of level set models is presented in Appendix A.

4.1 LS Speed Function Building Blocks

Given the definition

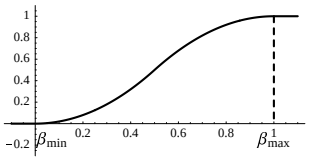
$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \dots) \equiv \boldsymbol{n} \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}, \tag{1}$$

the fundamental level set equation, Eq. (12), can be rewritten as

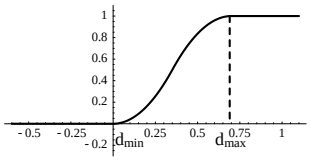
$$\frac{\partial \phi}{\partial t} = |\nabla \phi| \, \mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \dots) \tag{2}$$

where $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ and $\boldsymbol{n} \equiv -\nabla\phi/|\nabla\phi|$ are the velocity and normal vectors at $\boldsymbol{x}$ on the surface. We assume a positive-inside/negative-outside sign convention for $\phi(\boldsymbol{x}, t)$, *i.e.* $\boldsymbol{n}$ *points outwards*. Eq. (1) introduces the speed function $\mathcal{F}$, which is a user-defined scalar function that can depend on any number of variables including $\boldsymbol{x}$, $\boldsymbol{n}$, ϕ and its derivatives evaluated at $\boldsymbol{x}$, as well as a variety of external data inputs. $\mathcal{F}()$ is a *signed* scalar function that defines the motion (*i.e.* speed) of the level set surface in the direction of the local normal $\boldsymbol{n}$ at $\boldsymbol{x}$.

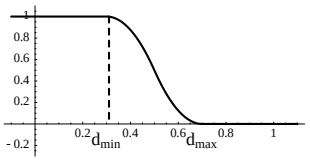
The speed function is usually based on a set of geometric measures of the implicit level set surface and data inputs. The challenge when working with level set methods is determining how to combine the building blocks to produce a local motion that creates a desired global or regional behavior of the surface. The general



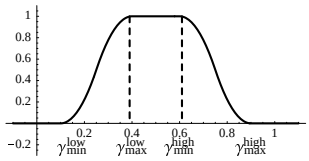
(a) Building-block, $P(\beta)$, for the ROI functions, *cf.* Eq. (4).



(b) Function, $\mathcal{D}_s(d)$, of distance to a surface, *cf.* Eq. (6b).



(c) Function, $\mathcal{D}_p(d)$, of distance to point set, *cf.* Eq. (6a).



(d) Function, $\mathcal{C}(\gamma)$, of geometric measure, γ, *cf.* Eq. (7).

Figure 4: Graph of region-of-influence (ROI) functions used to define the speed functions for our local level set operations, *cf.* Eq. (3).

structure for the speed functions used in our surface editing operators is

$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \mathcal{D}_q(d)\mathcal{C}(\gamma)\mathcal{G}(\gamma) \tag{3}$$

where $\mathcal{D}_q(d)$ is a distance-based cut-off function that depends on a distance measure d to a geometric structure q. $\mathcal{C}(\gamma)$ is a cut-off function that controls the contribution of $\mathcal{G}(\gamma)$ to the speed function. $\mathcal{G}(\gamma)$ is a function that depends on geometric measures γ derived from the level set surface, e.g. curvature. Thus, $\mathcal{D}_q(d)$ acts as a region-of-influence function that regionally constrains the LS calculation. $\mathcal{C}(\gamma)$ is a filter of the geometric measure and $\mathcal{G}(\gamma)$ provides the geometric contribution of the level set surface. In general γ is defined as zero, first, or second order measures of the LS surface.

4.2 Regionally Constraining LS Deformations

Most of our surface operators may be applied locally in a small user-defined region on the edited surface. In order to regionally restrict the deformation during the level set computation, a technique is needed for driving the value of $\mathcal{F}()$ to zero outside of the region. This is accomplished in three steps. The first step involves defining the region of influence (ROI), *i.e.* the region where $\mathcal{F}()$ should be non-zero. This is done by either the user interactively placing a 3D object around the region, or by automatically calculating a region from properties of the surface. Both cases involve defining a geometric structure that we refer to as a "region-of-influence (ROI) primitive". The nature of these primitives will vary for the different LS operations and will be explicitly defined in Section 5. The second step consists of calculating a distance measure to the ROI primitive. The final step involves defining a function that smoothly approaches zero at the boundary of the ROI.

We define a region-of-influence function $\mathcal{D}_q(d)$ in Eq. (3), where d is a distance measure from a point on the level set surface to the ROI primitive q. The functional behavior of $\mathcal{D}_q(d)$ clearly depends on the specific ROI primitive, q, but we found the following piecewise polynomial function to be useful as a common speed function building block:

$$P(\beta) = \begin{cases} 0 & \text{for } \beta \leq 0 \\ 2\beta^2 & \text{for } 0 < \beta \leq 0.5 \\ 1 - 2(\beta - 1)^2 & \text{for } 0.5 < \beta < 1 \\ 1 & \text{for } \beta \geq 1. \end{cases} \tag{4}$$

$P(\beta)$ and its derivatives are continuous and relatively inexpensive to compute. See Figure 4(a). Other continuous equations with the same basic shape would also be satisfactory. We then define

$$\mathcal{P}(d; d_{min}, d_{max}) \equiv P\left(\frac{d - d_{min}}{d_{max} - d_{min}}\right) \tag{5}$$

where d_{min} and d_{max} are user-defined parameters that define the limits and sharpness of the cut-off. Let us finally define the following region-of-influence functions

$$\mathcal{D}_p(d) = 1 - \mathcal{P}(d; d_{min}, d_{max}) \tag{6a}$$
$$\mathcal{D}_s(d) = \mathcal{P}(d; 0, d_{max}) \tag{6b}$$

for a point set, p, and a closed surface, s.

In Eq. (6a) d denotes the distance from a point on the level set surface to the closest point in the point set p. In Eq. (6b) d denotes a signed distance measure from a point on the level set surface to the implicit surface s. The signed distance measure does not necessarily have to be Euclidean distance - just a monotonic distance measure following the positive-inside/negative-outside convention. Note that $\mathcal{D}_p(d)$ is one when the shortest distance, d, to the point set is smaller than d_{min}, and decays smoothly to zero as d increases to d_{max}, after which it is zero. $\mathcal{D}_s(d)$, on the other hand, is zero everywhere outside, as well as on, the surface s ($d \leq 0$), but one inside when the distance measure d is larger than d_{max}.

An additional benefit of the region-of-influence functions is that they define the portion of the volume where the surface cannot move. We use this information to determine what voxels should be updated during the level set deformation, significantly lowering the amount of computation needed when performing editing operations. This technique allows our operators to be rapidly computed when modifying large models.

4.3 Limiting Geometric Property Values

We calculate a number of geometric properties from the level set surface. The zero order geometric property that we utilize is shortest distance from the level set surface to some ROI primitive. The first order property is the surface normal, $\boldsymbol{n} \equiv -\nabla\phi/|\nabla\phi|$. Second order information includes a variety of curvature measures of the LS surface. In Appendix B we outline a new numerical approach to deriving the mean, Gaussian and principle curvatures of a level set surface. Our scheme has numerical advantages relative to traditional central finite difference schemes for computing the second order derivatives. We found mean curvature to be the most useful second order measure [Evans and Spruck 1991] for our application.

Another desirable feature of our operators is that they allow the user to control the geometric properties of surface in the region being edited. This feature is implemented with another cut-off function, $\mathcal{C}()$, within the level set speed function. $\mathcal{C}()$ allows the user to slow and then stop the level set deformation as a particular surface property approaches a user-specified value. We reuse the cut-off function, Eq. (5), defined in the previous section, as a building block for $\mathcal{C}()$. We define

$$\mathcal{C}(\gamma) = \begin{cases} \mathcal{P}(\gamma; \gamma_{min}^{low}, \gamma_{max}^{low}) & \text{for } \gamma \leq \overline{\gamma} \\ 1 - \mathcal{P}(\gamma; \gamma_{min}^{high}, \gamma_{max}^{high}) & \text{for } \gamma > \overline{\gamma} \end{cases} \tag{7}$$

where $\overline{\gamma} \equiv (\gamma_{max}^{low} + \gamma_{min}^{high})/2$. The four parameters γ_{min}^{low}, γ_{max}^{low}, γ_{min}^{high}, and γ_{max}^{high} define respectively the upper and lower bounds of the cut-off function, see Figure 4(d).

4.4 Constraining the Direction of LS Motions

Another important feature of the level set framework is its ability to control the direction of the level set deformation. We are able

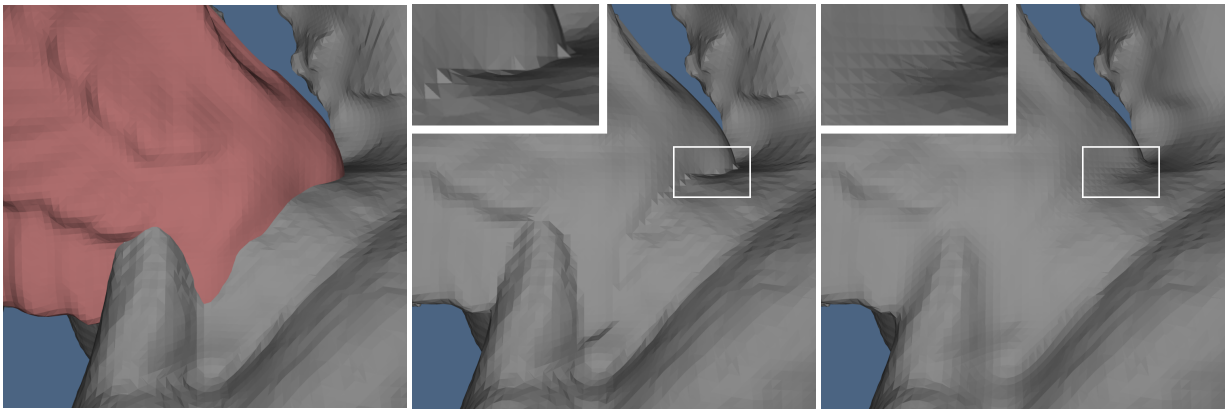


Figure 5: Left: Positioning the (red) wing model on the dragon model. Middle: The models are pasted together (CSG union operation), producing sharp, undesirable creases, a portion of which is expanded in the box. Right: Same region after automatic blending based on mean curvature. The blending is constrained to only move outwards. The models are rendered with flat-shading to highlight the details of the surface structure.

to restrict the motion of the surface to only add or remove material during the level set editing operations. At any point the level set surface can only move in the direction of the local surface normal. Hence, we can simply redefine the speed function as $\min(\mathcal{G}, 0)$ to remove material (inward motion only) and $\max(\mathcal{G}, 0)$ to add material (outward motion only). In the case of curvature driven speed functions this produces min/max flows [Sethian 1999]. Of course no restriction on the direction of the motion need be imposed.

5 Definition of Surface Editing Operators

Given the building blocks described in the previous section, the level set surface editing operators outlined in Figure 3 may be defined. We begin by defining the well-known CSG operations that are essential to most editing systems. We then define the new level set speed functions that implement our surface editing operators by combining the geometric measures with the region-of-influence and cut-off functions.

5.1 CSG Operations

Since level set models are volumetric, the constructive solid geometry (CSG) [Hoffmann 1989] operations of union, difference and intersection may be applied to them. This provides a straightforward approach to implementing copy, cut and paste operations on level set surfaces. In our level set framework, with a positive-inside/negative-outside sign convention for the distance volumes, these are implemented as min/max operations [Wang and Kaufman 1994] on the voxel values as summarized in Table 2. Any two closed surfaces represented as signed distance volumes can be used as either the main edited model or the cut/copy primitive. In our editing system the user is able to arbitrarily scale, translate and rotate the models before a CSG operation is performed.

Table 2: Implementation of CSG operations on two level set models, A and B, represented by distance volumes V_A and V_B with positive inside and negative outside values.

Action	CSG Operation	Implementation
Copy	Intersection, $A \cap B$	$\text{Min}(V_A, V_B)$
Paste	Union, $A \cup B$	$\text{Max}(V_A, V_B)$
Cut	Difference, $A - B$	$\text{Min}(V_A, -V_B)$

5.2 Automatic Localized LS Blending

The surface models produced by the CSG paste operation typically contain sharp and sometimes jagged creases at the intersection of the two surfaces. We can dramatically improve this region of the surface by applying an automatic localized blending. The method is automatic because it only requires the user to specify a few parameter values. It is localized because the blending operator is only applied near the surface intersection region. One possible solution to localizing the blending is to perform the deformation in regions near both of the input surfaces. However, this naive approach would result in blending the two surfaces in *all* regions of space where the surfaces come within a user-specified distance of each other, creating unwanted blends. A better solution, and the one we use, involves defining the region of influence based on the distance to the *intersection curve* shared by both input surfaces. A sampled representation of this curve is the set of voxels that contains a zero distance value (within some sub-voxel value ϵ) to both surfaces. We have found this approximate representation of the intersection curve as a point set to be sufficient for defining a shortest distance d for the region-of-influence function, $\mathcal{D}_p(d)$, *cf.* Eq. (3). Representing the intersection curve by a point set allows the curve to take an arbitrary form - it can even be composed of multiple curve segments without introducing any complications to the computational scheme.

The blending operator moves the surface in a direction that minimizes a curvature measure, $\mathcal{K}$, on the level set surface. This is obtained by making the speed function, $\mathcal{G}$, Eq. (3), proportional to $\mathcal{K}$, leading to the following blending speed function:

$$\mathcal{F}_{blend}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \alpha\, \mathcal{D}_p(d)\, \mathcal{C}(\mathcal{K})\, \mathcal{K} \tag{8}$$

where α is a user-defined positive scalar that controls the rate of convergence of the LS calculation, $\mathcal{D}_p(d)$ is defined in Eq. (6a) where d is the shortest distance from the level set surface to the intersection curve point set, and $\mathcal{C}(\mathcal{K})$ is given by Eq. (7) where $\mathcal{K}$ is one of the curvatures define in Appendix B. Through the functions $\mathcal{D}_p$ and $\mathcal{C}$ the user has full control over the region of influence of the blending (d_{min} and d_{max}) and the upper and lower curvature values of the blend (γ_{min}^{low}, γ_{max}^{low} and γ_{min}^{high}, γ_{max}^{high}). Furthermore we can control if the blend adds or removes material, or both as described in Section 4.4.

Automatic blending is demonstrated in Figure 5. A wing model is positioned relative to a dragon model. The two models are pasted together and automatic mean curvature-based blending is applied to smooth the creased intersection region.

5.3 Localized LS Smoothing/Sharpening

The smoothing operator smooths the level set surface in a user-specified region. This is accomplished by enclosing the region of interest by a geometric primitive. The "region-of-influence prim-

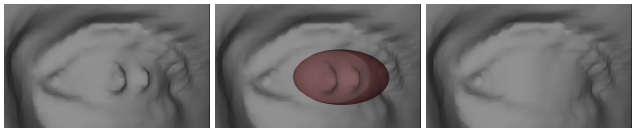


Figure 6: Regionally constrained smoothing. Left: Laser scan reconstruction with unwanted, pointed artifacts in the eye. Middle: Defining the region to be smoothed with a (red) superellipsoid. Right: Smoothing the surface within the superellipsoid. The surface is constrained to only move inwards.

itive" can be any closed surface for which we have signed inside/outside information, e.g. a level set surface or an implicit primitive. We use superellipsoids [Barr 1981] as a convenient ROI primitive, a flexible implicit primitive defined by two shape parameters. The surface is locally smoothed by applying motions in a direction that reduces the local curvature. This is accomplished by moving the level set surface in the direction of the local normal with a speed that is proportional to the curvature. Therefore the speed function for the smoothing operator is

$$\mathcal{F}_{smooth}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \alpha \mathcal{D}_s(d)\mathcal{C}(\mathcal{K})\mathcal{K}. \tag{9}$$

Here d denotes the signed value of the monotonic inside/outside function of the ROI primitive s evaluated at $\boldsymbol{x}$. As before, $\mathcal{D}_s(d)$ ensures that the speed function smoothly goes to zero as $\boldsymbol{x}$ approaches the boundary of the ROI primitive. $\mathcal{C}(\mathcal{K})$ limits the value of the surface's curvature within the ROI primitive.

Figure 6 demonstrates our smoothing operator applied to a laser scan reconstruction. Unwanted artifacts are removed from an eye by first placing a red superellipsoid around the region of interest. A smoothing operator constrained to only remove material is applied and the spiky artifacts are removed. Figure 7 demonstrates our smoothing operator applied to a preliminary 3D scan conversion of the Utah teapot. Unwanted artifacts are removed from the region where the spout meets the body of the teapot by first placing a superellipsoid around the region of interest. A smoothing operator constrained to only add material is applied and the crevices are removed. In our final, artificial smoothing example in Figure 8 a complex structure is completely smoothed away. This example illustrates that changes of topological genus and number of disconnected components are easily handled within a level set framework during smoothing.

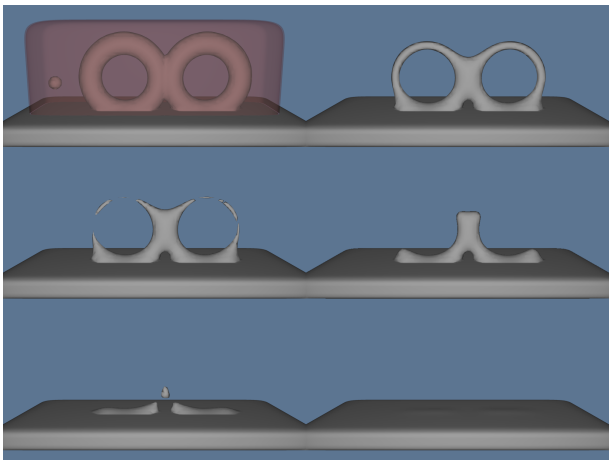


Figure 8: Changes in topological genus and the number of disconnected components are easily handled within a level set framework during smoothing. The superellipsoid defines the portion of the surface to be smoothed. The surface is constrained to move only inwards.

Figure 9: Left: Three types of single point attractions/repulsions using different ROI primitives and γ values. Right: Utah teapot embossed with 7862 points sampling the "SIGGRAPH 2002" logo.

We obtain a sharpening operator by simply inverting the sign of α in Eq. (9) and applying an upper cut-off to the curvature in $\mathcal{C}()$ in order to maintain numerical stability. The sharpening operator has been applied to the hair of the Greek bust in Figure 1.

5.4 Point Set Attraction and Embossing

We have developed an operator that attracts and repels the surface towards and away from a point set. These point sets can be samples of lines, curves, planes, patches and other geometric shapes, *e.g.* text. By placing the point sets near the surface, we are able to emboss the surface with the shape of the point set. Similar to the smoothing operator, the user encloses the region to be embossed with a ROI primitive *e.g.* a superellipsoid. The region-of-interest function for this operator is $\mathcal{D}_s(d)$, Eq. (6b).

First, assume that all of the attraction points are located outside the LS surface. $\boldsymbol{p}_i$ denotes the closest attraction point to $\boldsymbol{x}$, a point on the LS surface. Our operator only allows the LS surface to move towards $\boldsymbol{p}_i$ if the unit vector, $\boldsymbol{u}_i \equiv (\boldsymbol{p}_i - \boldsymbol{x})/|\boldsymbol{p}_i - \boldsymbol{x}|$, is pointing in the same direction as the local surface normal $\boldsymbol{n}$. Hence, the speed function should only be non-zero when $0 < \boldsymbol{n} \cdot \boldsymbol{u}_i \leq 1$. Since the sign of $\boldsymbol{n} \cdot \boldsymbol{u}_i$ is reversed if $\boldsymbol{p}_i$ is instead located inside the LS surface we simply require $\gamma = -\text{sign}[\phi(\boldsymbol{p}_i, t)]\boldsymbol{n} \cdot \boldsymbol{u}_i$ to be positive for any closest attraction point $\boldsymbol{p}_i$. This amounts to having only positive cut-off values for $\mathcal{C}(\gamma)$. Finally we let $\mathcal{G} = -\alpha\phi(\boldsymbol{p}_i, t)$ since this will guarantee that the LS surface will stop once it reaches $\boldsymbol{p}_i$. The following speed function implements the point set attraction operator:

$$\mathcal{F}_{point}(\boldsymbol{x}, \boldsymbol{n}, \phi) = -\alpha \mathcal{D}_s(d)\mathcal{C}(-\text{sign}[\phi(\boldsymbol{p}_i, t)]\boldsymbol{n} \cdot \boldsymbol{u}_i)\phi(\boldsymbol{p}_i, t), \tag{10}$$

where d is a signed distance measure to a ROI primitive evaluated at $\boldsymbol{x}$ on the LS surface, and $\boldsymbol{p}_i$ is the closest point in the set to $\boldsymbol{x}$. The shape of the primitive and the values of the four positive parameters in Eq. (7) define the footprint and sharpness of the embossing. See Figure 9, left. Point repulsion is obtained by making α negative. Note that Eq. (10) is just one example of many possible point set attraction speed functions.

In Figure 9, right, the Utah teapot is embossed with 7862 points that have been acquired by scanning an image of the SIGGRAPH 2002 logo and warping the points to fit the shape of the teapot.

5.5 Global Morphological Operators

The new level set operators presented above were designed to locally deform a level set surface. However, if the user wishes to perform a global smoothing of a level set surface, it is preferable to use an operator other than $\mathcal{F}_{smooth}$. For a global smoothing the level set propagation is computed on the whole volume, which can be slow for large volumes. However, in this case morphological opening and closing operators [Serra 1982] offer faster alternatives to global smoothing of level set surfaces. While we are not the first

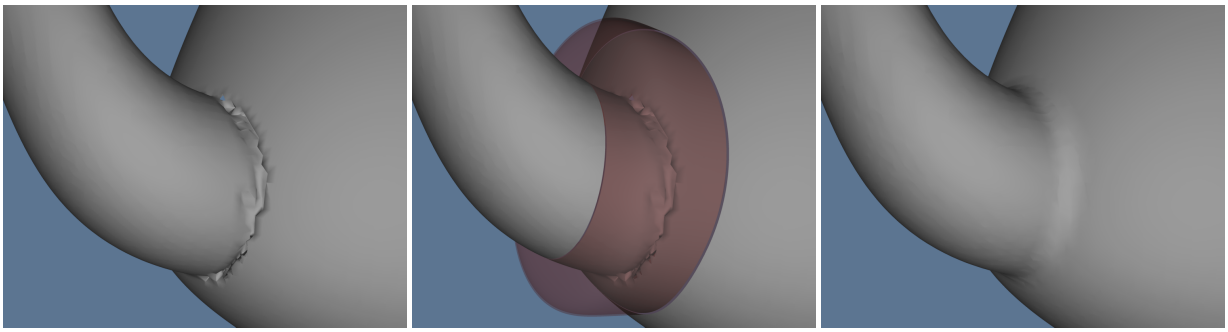


Figure 7: (left) Scan conversion errors near the teapot spout. (middle) Placing a (red) superellipsoid around the errors. (right) The errors are smoothed away in 15 seconds. The surface is constrained to only move outwards.

to explore morphological operators within a level set framework [Sapiro et al. 1993; Maragos 1996], we have implemented them and find them useful. Morphological openings and closings consist of two fundamental operators, dilations D_ω and erosions E_ω. Dilation creates an offset surface a distance ω outwards from the original surface, and erosion creates an offset surface a distance ω inwards from the original surface. The morphological opening operator O_ω is an erosion followed by a dilation, i.e. $\mathrm{O}_\omega = \mathrm{D}_\omega \circ \mathrm{E}_\omega$, which removes small pieces or thin appendages. A closing is defined as $\mathrm{C}_\omega = \mathrm{E}_\omega \circ \mathrm{D}_\omega$, and closes small gaps or holes within objects. Morphological operators may be implemented by solving a special form of the level set equation, the Eikonal equation, $\partial\phi/\partial t = \pm|\nabla\phi|$, up to a certain time t, utilizing Sethian's Fast Marching Method [Sethian 1996]. The value of t controls the offset distance from the original surface, $\phi(t = 0)$. Figure 10 contains a model from a laser scan reconstruction that has been smoothed with an opening operator with ω equal to 3.

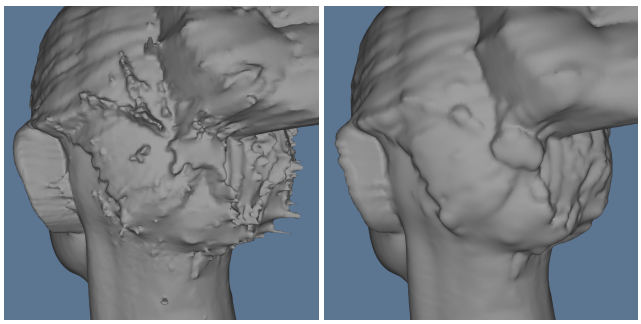


Figure 10: Applying a morphological opening to a laser scan reconstruction of a human head. The opening performs global smoothing by removing protruding structures smaller than a user-defined value.

5.6 Editing Session Details

Figure 11 contains a series of screen shots taken of our level set modeling program while constructing the two-headed winged dragon. The first shows the original dragon model loaded into the system. A cylindrical primitive is placed around its head and it is cut off. The model of the head is duplicated and the two heads are positioned relative to each other. Once the user is satisfied with their orientation, they are pasted together and an automatic blending is performed at the intersection seam. The combined double head model is positioned over the cropped neck of the dragon body. The double head is pasted and blended onto the body. The griffin model is loaded into the LS modeling system. A primitive is placed around one of its wings. The portion of the model within the primitive is copied, being stored in a buffer. Several cutting operations are used to trim the wing model (not shown). The double-headed dragon model is loaded, and the wing is positioned, pasted and blended onto it. A mirror copy of the wing model is created. It is also positioned, pasted and blended onto the other side of the double-headed dragon. We then added a loop onto the dragon's back as if designing a bracelet charm. This is accomplished by positioning, pasting, and blending a scan-converted supertoroid, producing the final model seen in the bottom right.

The Greek bust model was repaired by copying the nose from the human head model of Figure 10, and pasting and blending the copied model onto the broken nose. A piece from the right side of the bust was copied, mirrored, pasted and blended onto the left side of her face. Local smoothing operators were applied to various portions of her cheeks to clean minor cracks. Finally, the sharpening operator was applied within a user-defined region around her hair.

Table 3: Typical operator execution times on a R10K 250MHz MIPS processor.

Operation	Objects	sub-volume	Time
Paste	wing on dragon	$316 \times 172 \times 215$	33 sec.
Blend	wing on dragon	$82 \times 48 \times 63$	98 sec.
Smooth	teapot spout	$60 \times 55 \times 31$	15 sec.
Opening	human head	$256 \times 246 \times 193$	22 sec.
Emboss	single point	$21 \times 29 \times 29$	1.5 sec.

Table 4: Parameters used in examples. γ_{min}^{high} and γ_{min}^{high} are only used during sharpening. Their values are 0.8 and 0.9. No upper limit is placed on γ in the other examples.

Example	d_{min}	d_{max}	γ_{min}^{low}	γ_{max}^{low}
Wing Blending	7	9	0.04	0.06
Eye Smoothing	0.9	1	0.04	0.07
Spout Smoothing	0.9	1	0.1	0.13
Hair Sharpening	0.9	1	0.01	0.013
Teapot Embossing	0.9	1	0.8	0.9

6 Conclusion and Future Work

We have presented an approach to implementing surface editing operators within a level set framework. By developing a new set of level set speed functions automatic blending, localized smoothing and embossing may be performed on level set models. Additionally we have implemented morphological and volumetric CSG operators to fill out our modeling environment. All of the information needed to deform a level set surface is encapsulated in the speed function, providing a simple, unified computational framework. The level set framework offers several advantages. By construction, self-intersection cannot occur, which guarantees the generation of physically-realizable, simple, closed surfaces. Additionally, level set models easily change topological genus, and are free of the edge connectivity and mesh quality problems associated with mesh models.

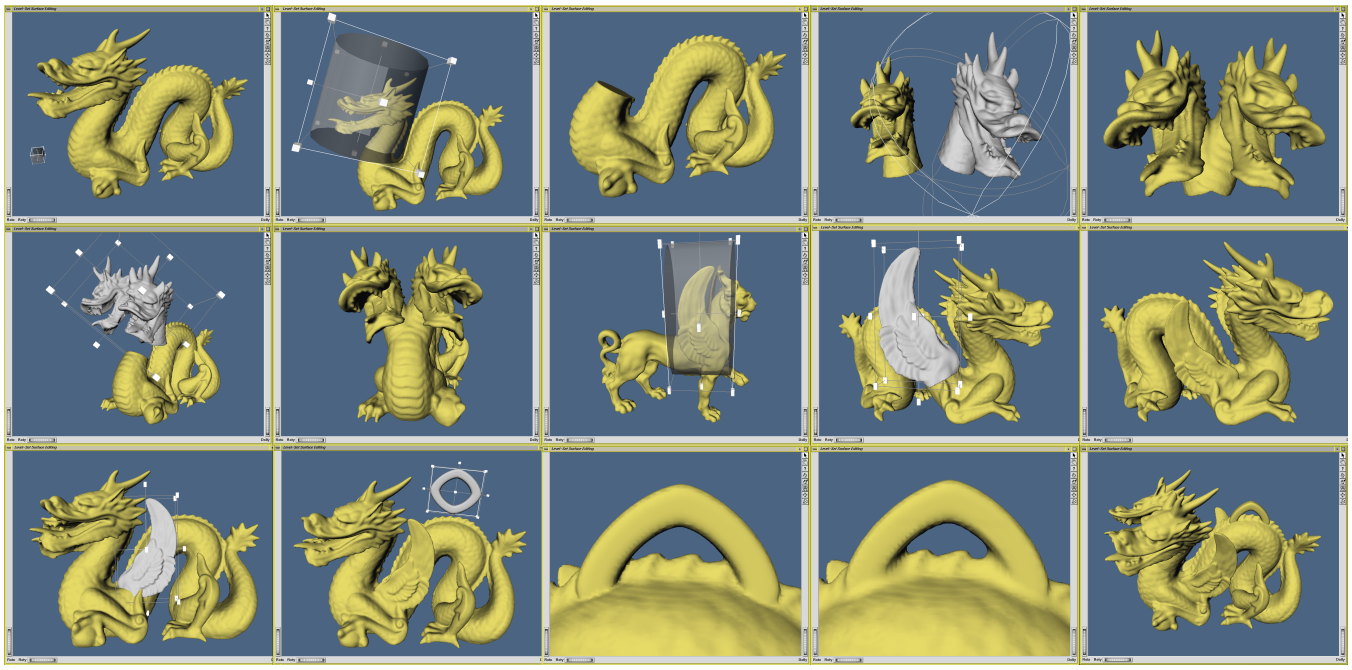


Figure 11: Series of operations used to create the winged two-headed dragon of Figure 1. First the head is cut off, pasted and blended back onto the body. Next a wing is copied from a different model and blended onto one side of the dragon. The same wing is then mirrored and blended onto the other side. Finally a scan converted supertoroid is blended onto the dragon's back to form the loop of a bracelet charm.

Several issues still must be addressed to improve our work. Currently level set implementations are based on uniform samplings of space, a fact that effectively limits the resolution of the objects that can be modeled. The development of adaptive level set methods would allow our operators to be applied to adaptive distance fields. It is possible to shorten the time needed to edit level set surfaces. Incrementally updating the mesh used to view the edited surface, utilizing direct volume rendering hardware, parallelizing the level set computations, and exploring multiresolution volumetric representations will lead to editing operations that require only a fraction of a second, instead of tens of seconds.

We have presented five example level set surface editing operators. Given the generality and flexibility of our framework many more can be developed. We intend to explore operators that utilize Gaussian and principal curvature, extend embossing to work directly with lines, curves and solid objects, and ones that may be utilized for general surface manipulations, such as dragging, warping, and sweeping.

7 Acknowledgements

We would like to thank Mathieu Desbrun for his helpful suggestions, and Katrine Museth and Cici Koenig for helping with the figures. The Greek bust and human head models were provided by Cyberware Inc. The dragon and griffin models were provided by the Stanford Computer Graphics Laboratory. The teapot model was provided by the University of Utah's Geometric Design and Computation Group. This work was financially supported by National Science Foundation grants ASC-89-20219, ACI-9982273 and ACI-0083287.

References

ADALSTEINSSON, D., AND SETHIAN, J. 1995. A fast level set method for propagating interfaces. *Journal of Computational Physics 118*, 269–277.

AMENTA, N., BERN, M., AND KAMVYSSELIS, M. 1998. A new voronoi-based surface reconstruction algorithm. In *Proc. SIGGRAPH '98*, 415–421.

BAJAJ, C., BERNARDINI, F., AND XU, G. 1995. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proc. SIGGRAPH '95*, 109–118.

BARR, A. 1981. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications 1*, 1, 11–23.

BIERMANN, H., KRISTJANSSON, D., AND ZORIN, D. 2001. Approximate Boolean operations on free-form solids. In *Proc. SIGGRAPH 2001*, 185–194.

BLOOMENTHAL, J., ET AL., Eds. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann, San Francisco.

BOUGUET, J.-Y., AND PERONA, P. 1999. 3D photography using shadow in dual space geometry. *International Journal of Computer Vision 35*, 2 (Nov/Dev), 129–149.

BREEN, D., AND WHITAKER, R. 2001. A level set approach for the metamorphosis of solid models. *IEEE Trans. on Visualization and Computer Graphics 7*, 2, 173–192.

BREEN, D., MAUCH, S., AND WHITAKER, R. 2000. 3D scan conversion of CSG models into distance, closest-point and colour volumes. In *Volume Graphics*, M. Chen, A. Kaufman, and R. Yagel, Eds. Springer, London, 135–158.

COHEN, E., RIESENFELD, R., AND ELBER, G. 2001. *Geometric Modeling with Splines*. AK Peters, Natick, MA.

CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH '96*, 303–312.

DESBRUN, M., AND GASCUEL, M. 1995. Animating soft substances with implicit surfaces. In *Proc. SIGGRAPH 95 Conference*, 287–290.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. SIGGRAPH '99*, 317–324.

DO CARMO, M. 1976. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, NJ.

EDELSBRUNNER, H., AND MÜCKE, E. 1994. Three-dimensional alpha shapes. *ACM Trans. on Graphics 13*, 1, 43–72.

EVANS, L., AND SPRUCK, J. 1991. Motion of level sets by mean curvature, I. *Journal of Differential Geometry 33*, 635–681.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proc. SIGGRAPH 2001*, 23–30.

FRISKEN, S., PERRY, R., ROCKWOOD, A., AND JONES, T. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *SIGGRAPH 2000 Proceedings*, 249–254.

GALYEAN, T., AND HUGHES, J. 1991. Sculpting: An interactive volumetric modeling technique. In *Proc. SIGGRAPH '91*, 267–274.

HOFFMANN, C. 1989. *Geometric and Solid Modeling.* Morgan Kaufmann, San Francisco.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. SIGGRAPH '98*, 105–114.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proc. SIGGRAPH 2001*, 57–66.

LAIDLAW, D., TRUMBORE, W., AND HUGHES, J. 1986. Constructive solid geometry for polyhedral objects. In *Proc. SIGGRAPH '86*, 161–170.

LORENSEN, W., AND CLINE, H. 1987. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH '87*, 163–169.

MALLADI, R., SETHIAN, J., AND VEMURI, B. 1995. Shape modeling with front propagation: A level set approach. *IEEE Trans. on Pattern Analysis and Machine Intelligence 17*, 2, 158–175.

MARAGOS, P. 1996. Differential morphology and image processing. *IEEE Trans. on Image Processing 5*, 6 (June), 922–937.

OSHER, S., AND FEDKIW, R. 2001. Level set methods: An overview and some recent results. *Journal of Computational Physics 169*, 475–502.

OSHER, S., AND SETHIAN, J. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics 79*, 12–49.

PENG, D., MERRIMAN, B., OSHER, S., ZHAO, H.-K., AND KANG, M. 1999. A PDE-based fast local level set method. *Journal of Computational Physics 155*, 410–438.

PERRY, R., AND FRISKEN, S. 2001. Kizamu: A system for sculpting digital characters. In *Proc. SIGGRAPH 2001*, 47–56.

REQUICHA, A., AND VOELCKER, H. 1985. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE 73*, 1, 30–44.

RUDIN, L., OSHER, S., AND FATEMI, C. 1992. Nonlinear total variation based noise removal algorithms. *Physica D 60*, 259–268.

SAPIRO, G., KIMMEL, R., SHAKED, D., KIMIA, B., AND BRUCKSTEIN, A. 1993. Implementing continuous-scale morphology via curve evolution. *Pattern Recognition 26*, 9, 1363–1372.

SAPIRO, G. 2001. *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press, Cambridge, UK.

SERRA, J. 1982. *Image Analysis and Mathematical Morphology*. Academic Press, London.

SETHIAN, J. 1996. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Science*, vol. 93, 1591–1595.

SETHIAN, J. 1999. *Level Set Methods and Fast Marching Methods*, second ed. Cambridge University Press, Cambridge, UK.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proc. SIGGRAPH '95*, 351–358.

TSITSIKLIS, J. 1995. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control 40*, 9, 1528–1538.

WANG, S., AND KAUFMAN, A. 1994. Volume-sampled 3D modeling. *IEEE Computer Graphics and Applications 14*, 5 (September), 26–32.

WANG, S., AND KAUFMAN, A. 1995. Volume sculpting. In *Proc. Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 151–156.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *Proc. SIGGRAPH '94*, 247–256.

WHITAKER, R., AND XUE, X. 2001. Variable-conductance, level-set curvature for image denoising. In *Proc. IEEE International Conference on Image Processing*, 142–145.

WHITAKER, R., BREEN, D., MUSETH, K., AND SONI, N. 2001. Segmentation of biological datasets using a level-set framework. In *Volume Graphics 2001*, M. Chen and A. Kaufman, Eds. Springer, Vienna, 249–263.

WHITAKER, R. 1998. A level-set approach to 3D reconstruction from range data. *International Journal of Computer Vision 29*, 3, 203–231.

WYVILL, B., GALIN, E., AND GUY, A. 1999. Extending the CSG tree. warping, blending and Boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (June), 149–158.

ZHAO, H.-K., OSHER, S., AND FEDKIW, R. 2001. Fast surface reconstruction using the level set method. In *Proc. 1st IEEE Workshop on Variational and Level Set Methods*, 194–202.

A Level Set Models

A deformable (*i.e.* time-dependent) surface, $\mathcal{S}(t)$, is implicitly represented as an iso-surface of a time-varying[1] scalar function, $\phi(\boldsymbol{x}, t)$, embedded in 3D, *i.e.*

$$\mathcal{S}(t) = \{\boldsymbol{x}(t) \mid \phi(\boldsymbol{x}(t), t) = k\}, \tag{11}$$

where $k \in \Re$ is the iso-value, $t \in \Re^+$ is time, and $\boldsymbol{x}(t) \in \Re^3$ is a point in space on the iso-surface. It might seem inefficient to implicitly represent a surface with a 3D scalar function; however the higher dimensionality of the representation provides one of the major advantages of the LS method: the flexible handling of changes in the topology of the deformable surface. This implies that LS surfaces can easily represent complicated surface shapes that can, form holes, split to form multiple objects, or merge with other objects to form a single structure.

The fundamental level set equation of motion for $\phi(\boldsymbol{x}(t), t)$ is derived by differentiating both sides of Eq. (11) with respect to time t, and applying the chain rule giving:

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}, \tag{12}$$

where $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ denotes the speed vectors of the level set surface. A number of numerical techniques by [Osher and Sethian 1988; Adalsteinsson and Sethian 1995] make the initial value problem of Eq. (12) computationally feasible. A complete discussion of the details of the level set method is beyond the scope of this paper. We instead refer the interested reader to [Sethian 1999; Osher and Fedkiw 2001]. However, we will briefly mention two of the most important techniques: the first is the so called "up-wind scheme" which addresses the problem of overshooting when trying to solve Eq. (12) by a simple finite forward difference scheme. The second is related to the fact that one is typically only interested in a single solution to Eq. (12), say the $k = 0$ level set. This implies that the evaluation of ϕ is important only in the vicinity of that level set. This forms the basis for "narrow-band" schemes [Adalsteinsson and Sethian 1995; Whitaker 1998; Peng et al. 1999] that solve Eq. (12) in a narrow band of voxels containing the surface. The "up-wind scheme" makes the level set method numerically robust, and the "narrow-band scheme" makes its computational complexity proportional to the level set's surface area rather than the size of the volume in which it is embedded.

[1] Our work uses the dynamic level set equation, which is more flexible than the corresponding stationary equation, $\phi(\boldsymbol{x}) = k(t)$, see [Sethian 1999] for more details.

B Curvature of Level Set Surfaces

The principle curvatures and principle directions are the eigenvalues and eigenvectors of the *shape matrix* [do Carmo 1976]. For an implicit surface, the shape matrix is the derivative of the normalized gradient (surface normals) projected onto the tangent plane of the surface. If we let the normals be $\boldsymbol{n} = \nabla\phi/|\nabla\phi|$, the derivative of this is the 3×3 matrix

$$\boldsymbol{N} = \begin{pmatrix} \frac{\partial \boldsymbol{n}}{\partial x} & \frac{\partial \boldsymbol{n}}{\partial y} & \frac{\partial \boldsymbol{n}}{\partial z} \end{pmatrix}^T. \tag{13}$$

The projection of this derivative matrix onto the tangent plane gives the shape matrix [do Carmo 1976] $\boldsymbol{B} = \boldsymbol{N}(I - \boldsymbol{n} \otimes \boldsymbol{n})$, where $\otimes$ is the exterior product. The eigenvalues of the matrix $\boldsymbol{B}$ are k_1, k_2 and zero, and the eigenvectors are the principle directions and the normal, respectively. Because the third eigenvalue is zero, we can compute k_1, k_2 and various differential invariants directly from the invariants of $\boldsymbol{B}$. Thus the weighted curvature flow is computing from $\boldsymbol{B}$ using the identities $D = ||\boldsymbol{B}||_2$, $H = \mathrm{Tr}(\boldsymbol{B})/2$, and $K = 2H^2 - D^2/2$. The principle curvatures are calculated by solving the quadratic

$$k_{1,2} = H \pm \sqrt{\frac{D^2}{2} - H^2}. \tag{14}$$

In many circumstances, the curvature term, which is a kind of directional diffusion that does not suffer from overshooting, can be computed directly from first- and second-order derivatives of ϕ using central difference schemes. However, we have found that central differences do introduce instabilities when computing flows that rely on quantities other than the mean curvature. Therefore we use the method of *differences of normals* [Rudin et al. 1992; Whitaker and Xue 2001] in lieu of central differences. The strategy is to compute normalized gradients at staggered grid points and take the difference of these staggered normals to get centrally located approximations to $\boldsymbol{N}$. The shape matrix $\boldsymbol{B}$ is computed with gradient estimates from central differences. The resulting curvatures are treated as speed functions (motion in the normal direction), and the associated gradient magnitude is computed using the up-wind scheme.

Module 3

PDE Applications and Implementation

Image Inpainting: An Overview

Guillermo Sapiro

University of Minnesota

guille@ece.umn.edu

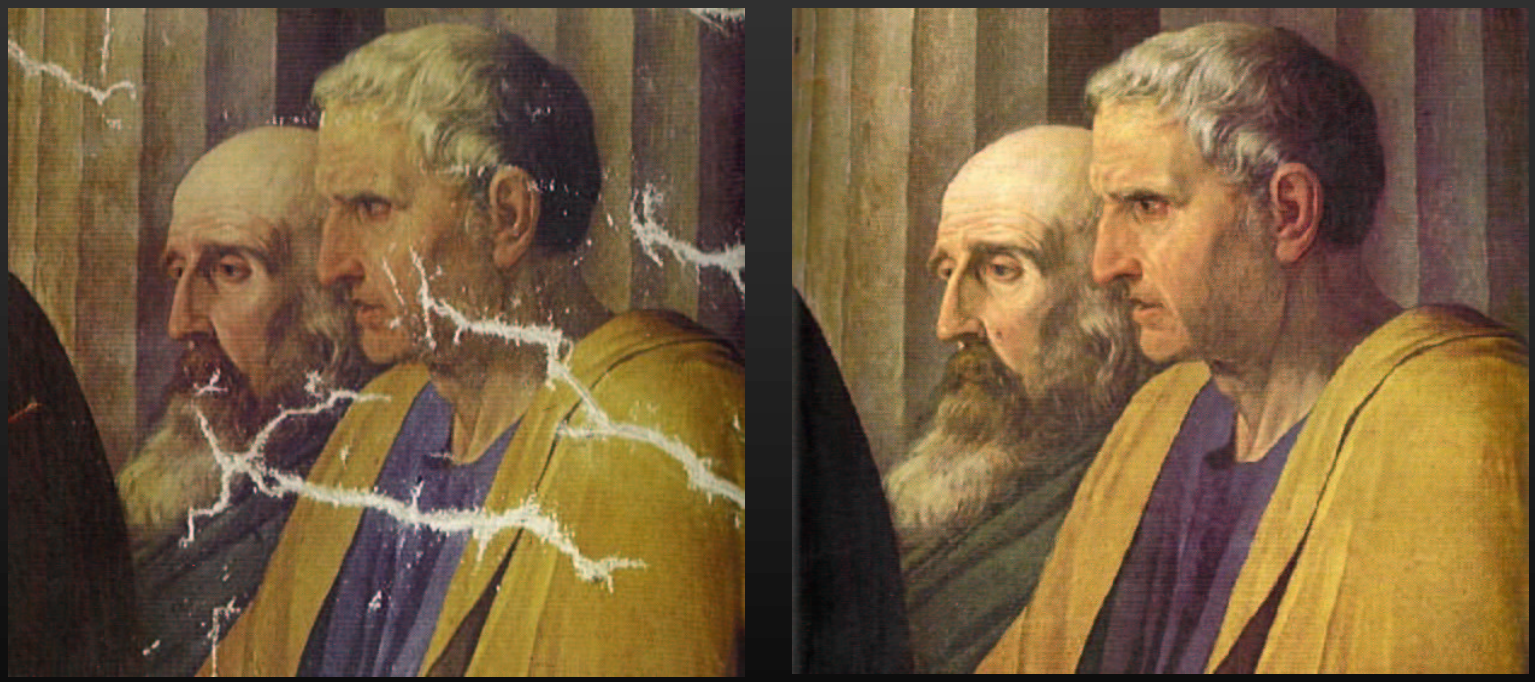
Supported by NSF, ONR, PECASE, CAREER, NIH

Overview

- Goal
- Related work
- Inpainting
- Filling-in

What is inpainting?

- Modifying an image in a non-detectable form



Detail of "Cornelia, Mother of the Gracchi" by J. Suvee (Louvre).
Taken from Emile-Male “The Restorer’s Handbook of easel painting”.

Another example

From Geary Gallery

Real world example: Photo restoration

- Restorations courtesy of Photo Imaging Studio, Image Enigma, Alleycat Designs

Real world example: Object removal

- From D. King, “The Commissar vanishes”.

Real world example: Object removal

Lenin and friend Trotsky

Where is Trotsky?

- From www.newseum.org

Real world example: Object removal and missing information

- From ProSpec-UK.

The goal

Related work: Films

- e.g. Kokaram et al.

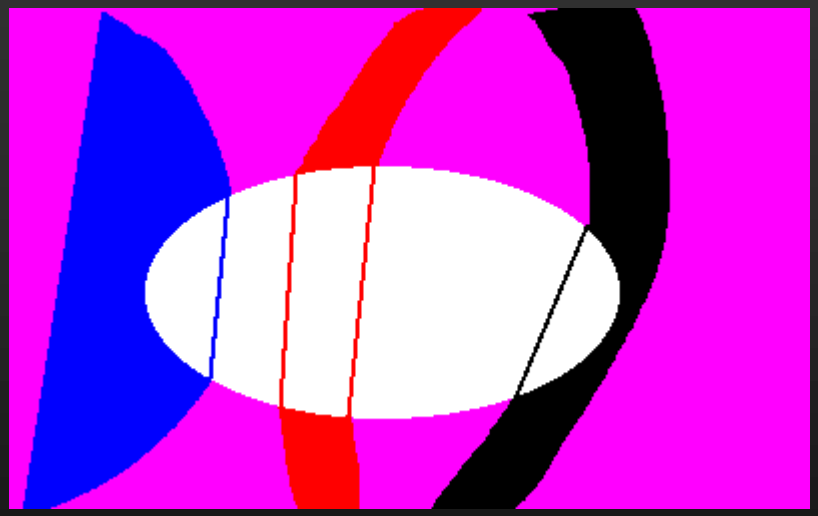
- Doesn’t work for stills or static objects

Related work: Texture synthesis

- Hirani, Efros, Heeger, DeBonet, Simoncelli, Zhu, etc.
- Not practical for rich regions
- Not designed for structured regions
- “Copy” information instead of “see and interpolate”

Related work: Disocclusion

- Masnou-Morel, Nitzberg-Mumford, etc.



- Limitations: Topology, angles

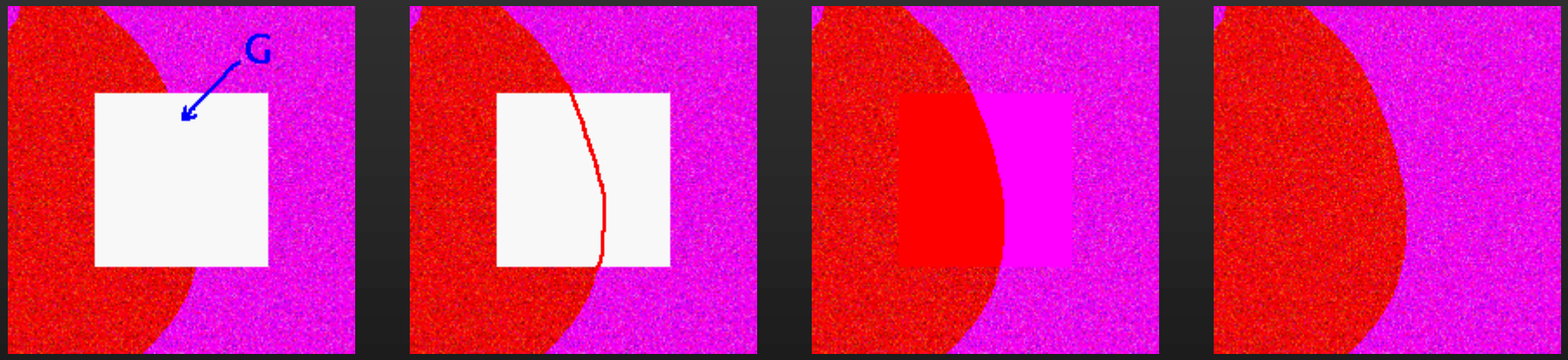
See also Jacobs, Basri, Zucker, etc, and Chan-Shen ‘00, Zhu-Mumford

Our Contribution

- User only selects region to inpaint
- Rich background and topology not an issue
- Less than 30 seconds on a PC

How conservators inpaint

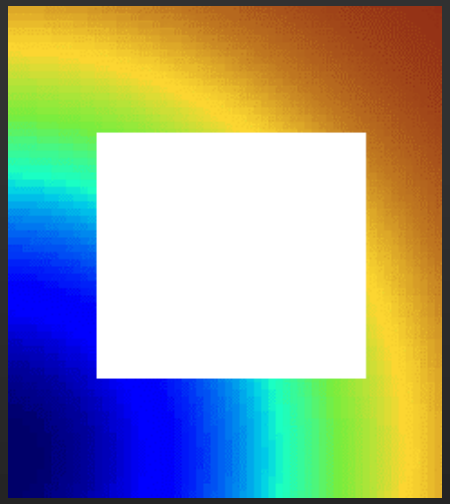
- Minneapolis Institute of Art

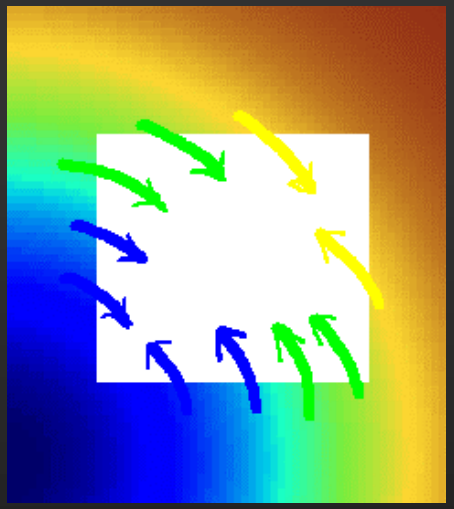


Approach 1

Bertalmio, Sapiro, Caselles, Ballester,
SIGGRAPH 2000

Automatic digital inpainting

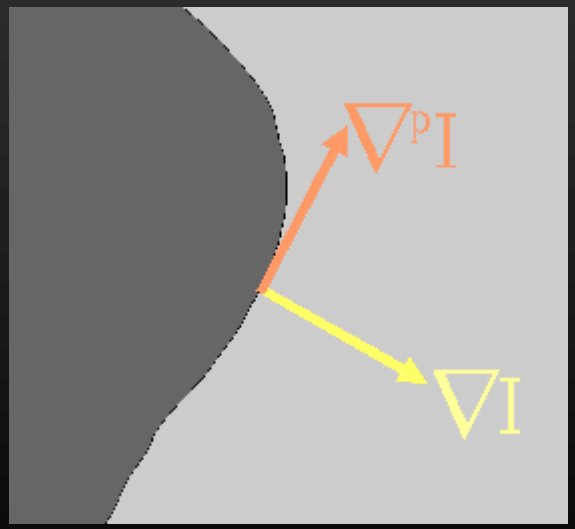


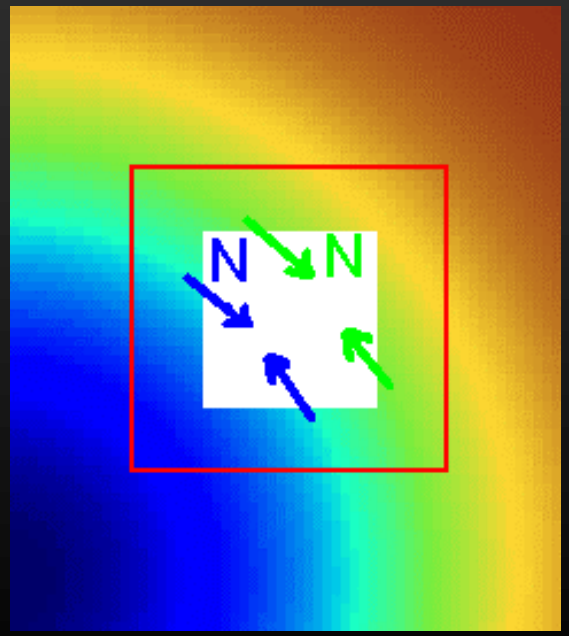


- Propagate information $\nabla L \bullet \vec{N} = 0$
- Evolutionary form $\frac{\partial I}{\partial t} = \nabla L \bullet \vec{N}$

Digital inpainting (cont'd)

- L = smoothness estimator (Laplacian)
- N = isophote direction (time variant)

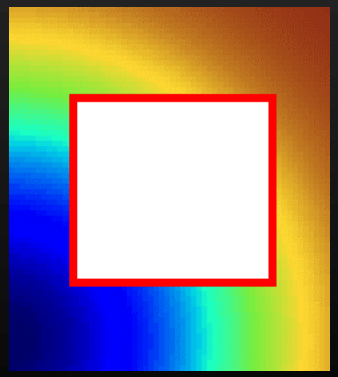




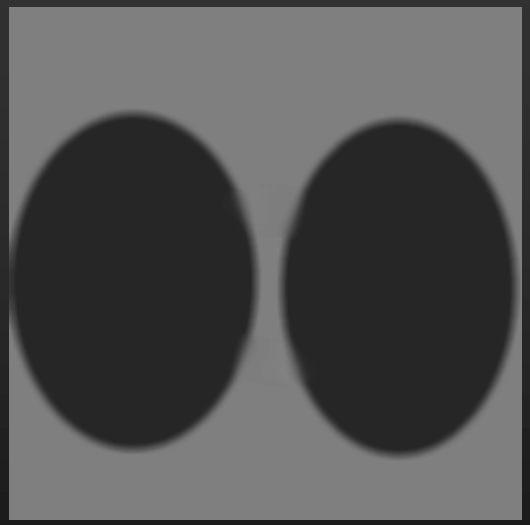
The equation

$$\frac{\partial I}{\partial t} = \nabla(\Delta I) \bullet \nabla^{\perp} I$$

- **Plus numerical schemes (Osher)**
- **Boundary conditions**
  - Gray values (in a band)
  - Directions (in a band)

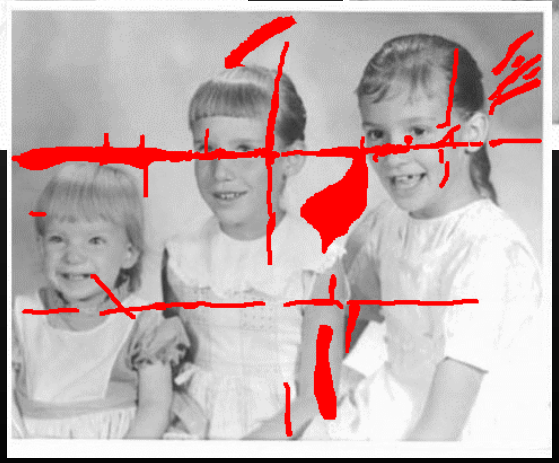


Example



Example: Text removal

Example: Photo restoration



Example: Special effects

Example: Scratch removal

Example: The evolution

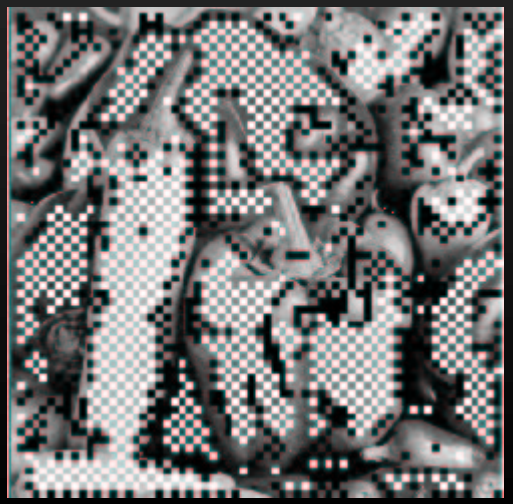
Automatic image inpainting/interpolation for compression and wireless transmission

(Rane-Sapiro-Bertalmio) JPEG and/or JPEG-2000 compatible

Transmitted

Automatic reconstruction

Transmitted



Automatic reconstruction

Approach 1: Concluding remarks

- **Technique imitates professionals**
- **Key concepts**
  - Information propagation
  - Both gray values and directions are needed
  - Use a band surrounding the region
- **Sharp results**
- **Low complexity**
- **Texture is not reproduced**

Concluding remarks (cont.)

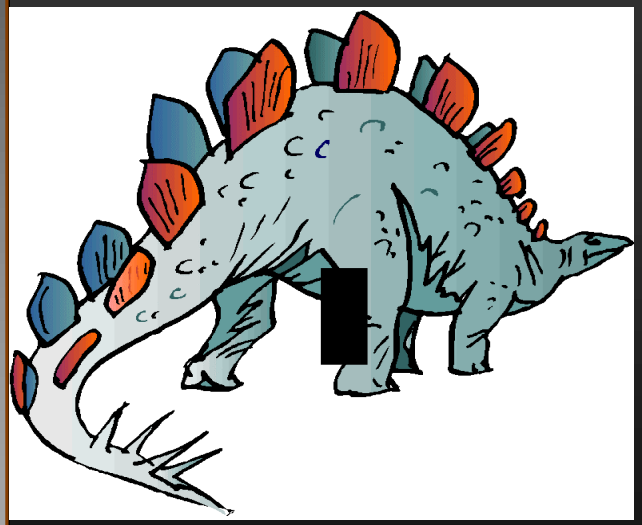
- Connected to fluid dynamics (see Bertalmio-Bertozzi-Sapiro CVPR 2001)
- Opens then door to high order PDE's
- Extended to a variational formulation: Approach 2...

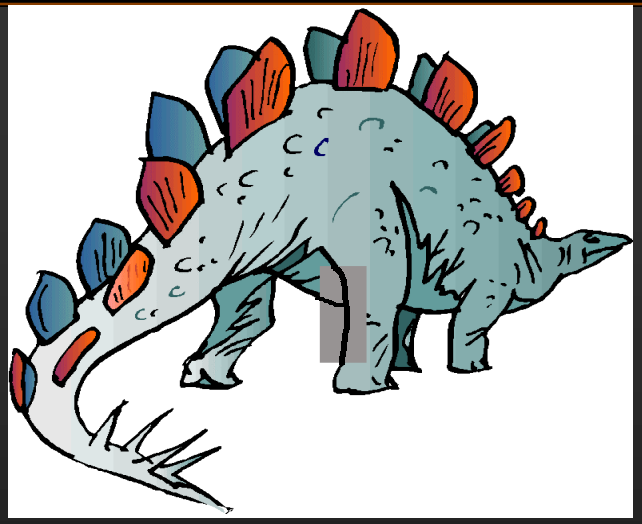
Approach 2

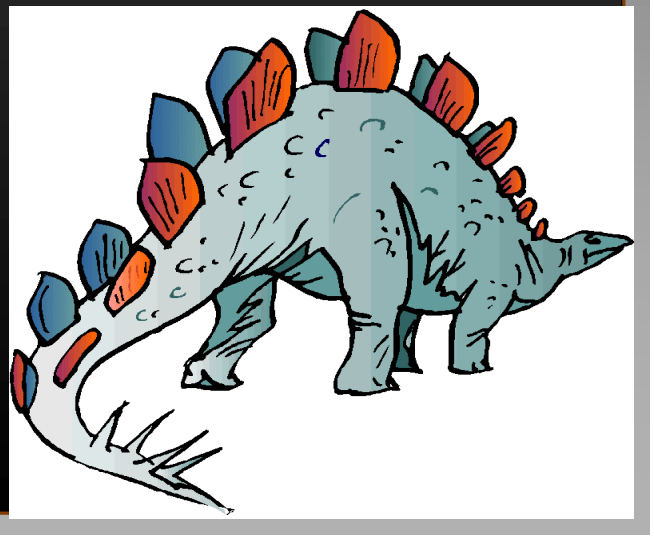
C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera, IMA Report 2000, IEEE Trans. IP 2001

How conservators fill-in

(Minneapolis Institute of Art)

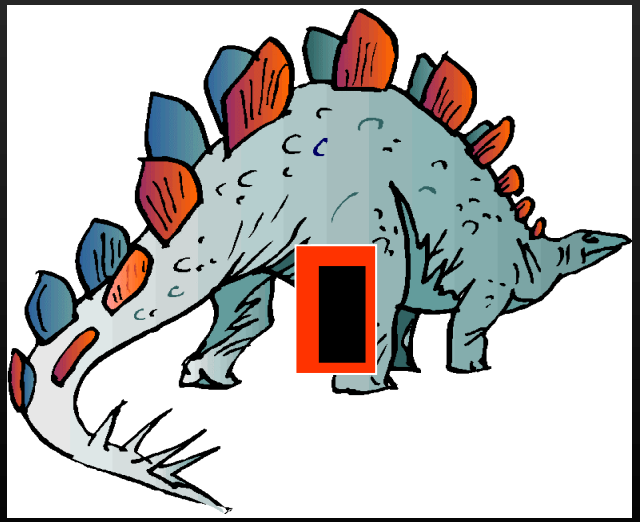




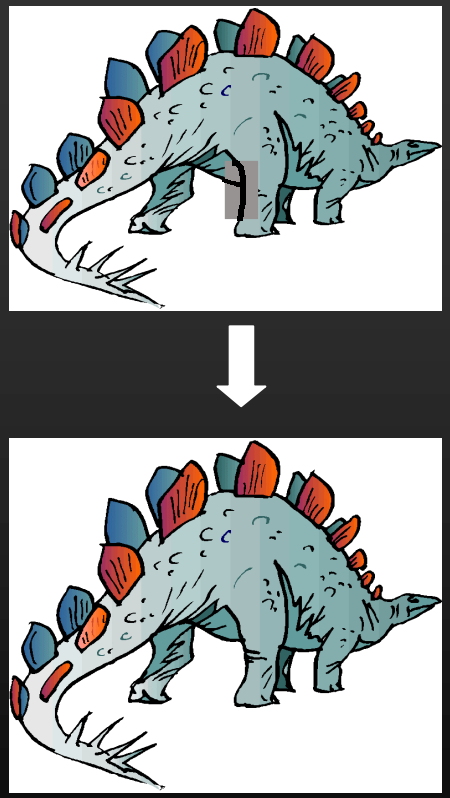


Our approach

- Jointly continue/interpolate level-lines (geometry) and gray values (photometry) in a smooth fashion



Interpolate the gray values given the edges



$$\theta = \text{normalized gradient} \Rightarrow \theta \bullet \nabla I = \|\nabla I\|$$

$$\min(I) \int_{\Omega \cup Band} (\|\nabla I\| - \theta \bullet \nabla I)\, d\Omega$$

$$\frac{\partial I}{\partial t} = div\left(\frac{\nabla I}{\|\nabla I\|}\right) - div(\theta)$$

Theorem: The minimizer exists in BV space

Example



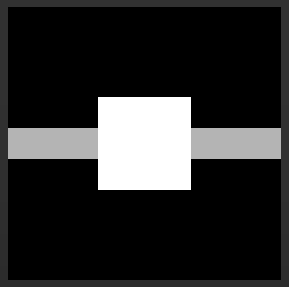
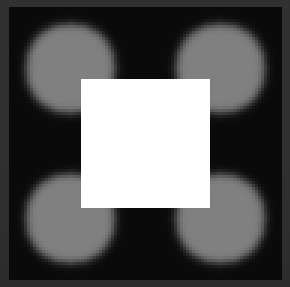
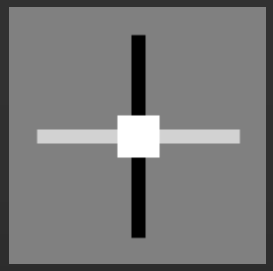
The full functional

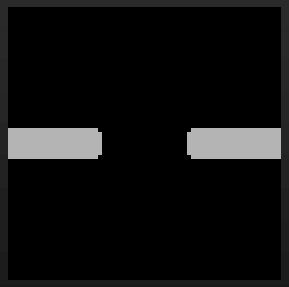
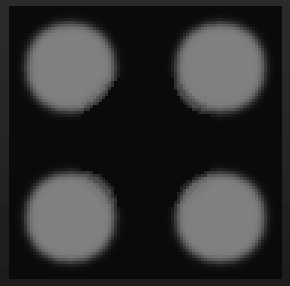
$$\min_{(u,\theta)} \int_{\Omega \cup Band} div(\theta)^p (a + b\|\nabla G * u\|) + c(\|\nabla u\| - \theta \bullet \nabla u)$$

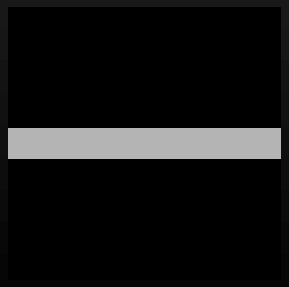
- Solved via E-L: Coupled 2nd order PDE's
- Implicit discretization used
- Connected to Euler's elastica (Mumford)

- **Theorem:** For p>1 the minimizer exists

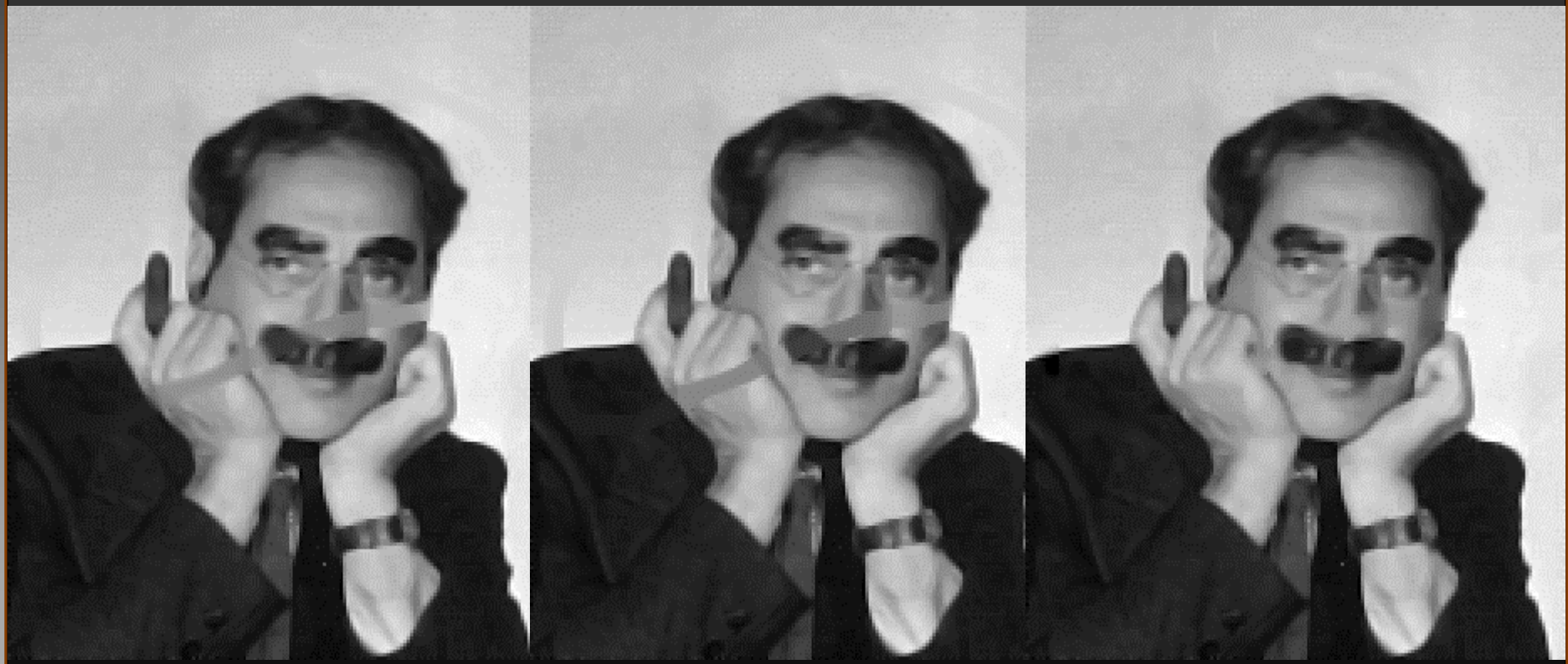
Examples

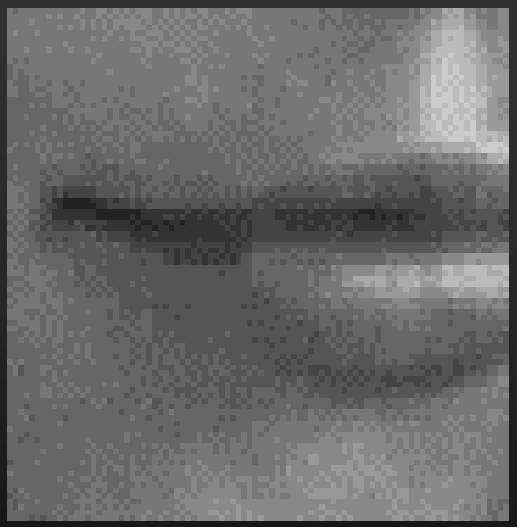




No edge information (just gray values, TV)



Examples

Examples



Examples


Il Castello di Miramare e i
suo parco sorsero sul promo
torio roccioso d'origine ca
sica di Grignano, per volon
dell' arciduca Ferdinando M
ssimiliano d'Asburgo (1832-
fratello minore dell'impera
austriaco Francesco Giusepp
Progettato nel 1856 da Carl
terminato nell'aspetto este
nel 1860. La sistemazione d
e la decorazione interna, o
e Julius Hofmann, furono ul
dopo la partenza di Massimi
per il Messico nel 1864. No
imperatore del Messico, Mas
venne fucilato a Queretaro
1867. Tra i pochissimi esem
di una dimora nobile conser
senza rifacimenti, il Caste

Approach 2: Concluding remarks

- **Technique imitates professionals**
- **Key concepts**
  - Information propagation
  - Both gray values and directions are needed
  - Use a band surrounding the region
- **Sharp results**
- **Low complexity**
- **Texture is not (yet) reproduced** **(Zhu et al, Acton et al.)**

Acknowledgments

- Institute Henri Poincare in Paris, France.
- T. Robbins, E. Buschor, S. Betelú, S. Osher, E. Simoncelli, A. Bertozzi, T. Chan, C. Kenney, P. L. Lions, J. M. Morel, J. Shen.
- Supported by ONR-Math, ONR Young Investigator Award, Presidential Early Career Awards for Scientists and Engineers, NSF CAREER Award, NSF-LIS, Europen project on viscosity solutions, and IIE-Uruguay.

The end

Thank you

Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting

M. Bertalmio
Computer Eng. Dept.
University Pompeu Fabra
08003 Barcelona, SPAIN

A. L. Bertozzi
Mathematics Dept.
Duke University
Durham, NC 27708

G. Sapiro
Elec. & Comp. Eng. Dept.
Univ. of Minnesota
Minneapolis, MN 55455

Abstract

Image inpainting involves filling in part of an image or video using information from the surrounding area. Applications include the restoration of damaged photographs and movies and the removal of selected objects. In this paper, we introduce a class of automated methods for digital inpainting. The approach uses ideas from classical fluid dynamics to propagate isophote lines continuously from the exterior into the region to be inpainted. The main idea is to think of the image intensity as a 'stream function' for a two-dimensional incompressible flow. The Laplacian of the image intensity plays the role of the vorticity of the fluid; it is transported into the region to be inpainted by a vector field defined by the stream function. The resulting algorithm is designed to continue isophotes while matching gradient vectors at the boundary of the inpainting region. The method is directly based on the Navier-Stokes equations for fluid dynamics, which has the immediate advantage of well-developed theoretical and numerical results. This is a new approach for introducing ideas from computational fluid dynamics into problems in computer vision and image analysis.

1. Introduction

Image *inpainting* [2, 10, 20, 38] is the process of filling in missing data in a designated region of a still or video image. Applications range from removing objects from a scene to re-touching damaged paintings and photographs. The goal is to produce a revised image in which the inpainted region is seamlessly merged into the image in a way that is not detectable by a typical viewer. Traditionally, inpainting has been done by professional artists. For photography and film, inpainting is used to revert deterioration (e.g., cracks in photographs or scratches and dust spots in film), or to add or remove elements (e.g., removal of stamped date and red-eye from photographs, the infamous "airbrushing" of political enemies [20]). A current active area of research is to automate digital techniques for inpainting [2, 3, 16, 21, 22].

In this paper, we introduce a novel algorithm for digital inpainting of still images that attempts to replicate the basic techniques used by professional restorators. Our algorithm, motivated by a method proposed in [2], involves a direct solution of the Navier-Stokes equations for an incompressible fluid. The image intensity function plays the role of the stream function whose isophote lines define streamlines of the flow. After the user selects the regions to be restored, the algorithm automatically transports information into the inpainting region. The fill-in is done in such a way that isophote lines arriving at the region's boundaries are completed inside. The technique introduced here does not require the user to specify where the novel information comes from. This is done automatically (and in a fast way), thereby allowing for simultaneously fill-in of multiple regions containing completely different structures and surrounding backgrounds. In addition, no limitations are imposed on the topology of the region to be inpainted. The only user interaction required by the algorithm is to mark the regions to be inpainted. Since our inpainting algorithm is designed for both restoration of damaged photographs and for removal of undesired objects on the image, the regions to be inpainted must be marked by the user. Our method inherits a mathematical theory already developed for the fluid equations, including well-posedness and the design of efficient convergent numerical methods.

1.1. Prior Work

First note that image denoising is different to filling-in, since the regions of missing data are usually large. That is, regions occupied by top to bottom scratches along several film frames, long cracks in photographs, superimposed large fonts, and so on, are of significantly larger size than the type of noise treated by common image enhancement algorithms. In addition, in common image enhancement applications, the pixels contain both information about the real data and the noise (e.g., image plus noise for additive noise), while in our application there is no significant information in the region to be inpainted.

A very active area related to our work is the restoration of damaged films. The basic idea is to use information from past and future frames to restore the current one, e.g., [16, 22]. Of course, this general approach cannot be

used when dealing with still images. In addition, it can not deal with movies where the region to be inpainted is static with respect to its background (e.g., a logo on a shirt), since consecutive frames do not provide new information. This is also the case when the region to be inpainted occupies a large number of frames. Another area related to our work is texture synthesis, in which a texture is selected and synthesized inside the region to be filled-in (the hole) [9, 13, 15, 34]. These algorithms often require the user to select the texture and are not often well-designed to fill in structure from boundary data.

The closest methods to our approach are the fundamental works on disocclusion and line continuation. A pioneering contribution in this area is described in [28]. The authors presented a technique for removing occlusions with the goal of image segmentation. Since the region to be filled-in can be considered as occluding objects, removing occlusions is analogous to image inpainting. The basic idea suggested by the authors is to connect T-junctions at the occluding boundaries of objects with elastica minimizing curves. The technique was primarily developed for simple images obtained from a segmentation, with only a few objects with constant gray-levels. Thus, they ended up by connecting T-junctions at the same gray level. (Other researchers, e.g., D. Jacobs, R. Basri, and S. Zucker, have followed this interesting research area, mainly developing techniques for smooth curve continuation.)

Masnou and Morel [25, 26] recently extended these ideas, presenting a formal variational formulation for disocclusion and a particular practical implementation. The algorithm uses geodesic curves to join the isophotes arriving at the boundary of the region to be inpainted. The inpainting regions require a simple topology. In addition, the angle, with which the level lines arrive at the boundary of the holes, is not (well) preserved, and the algorithm uses straight lines to join equal gray value pixels.

1.2. Image inpainting along isophotes

Recently, the concept of smooth continuation of information in the level-lines direction has been addressed in [2]. This is the work, as we will see below, that we follow to make the connection with fluid dynamics and the Navier-Stokes equations. The proposed algorithm propagates the image Laplacian in the level-lines (isophotes) direction. The algorithm attempts to imitate basic approaches used by professional restorators. The algorithm also introduces the importance of propagating both the gradient direction (geometry) and gray-values (photometry) of the image in a band surrounding the hole to be filled-in. Some of the ideas of [2] where adopted in [1], while deviating from the particular model in order to be able to define a formal variational approach to the filling-in/inpainting problem.

The work in [2] inspired a very elegant approach to the filling-in problem recently reported in [7] (this work was performed independently of the one reported in [1]). The authors present a clear and intuitive axiomatic approach to the problem. The main algorithm they propose is to minimize the Total Variation (TV) [33], of the image inside the hole (they also use, as proposed in [1, 2] and here, a band surrounding the region). As in the work of Masnou and Morel, their interpolation is limited to creating straight isophotes, not necessarily smoothly continued from the hole boundary, and mainly is developed (as the authors clearly state) for small holes. Although straight connections give visually pleasant results for small holes, it is important to develop a theory that permits interpolation of level lines across large gaps, where connecting with straight lines will be unpleasant even for simple images. In order to obtain such a smooth interpolation and continuation of isophotes, it is necessary to go into high-order Partial Differential Equations (PDE's) or systems of PDE's, as done in [1, 2] and here (the authors of [7] have also recently introduced higher order models). Research in perception, from the Gestalt to more recent work (e.g. [31]) supports the idea of performing a smooth continuation of the angle of arrival of the level lines at the gap.

The paper [2] proposes an algorithm designed to project the gradient of the smoothness of the image intensity in the direction of the isophotes. The resulting scheme is a discrete approximation of the PDE

$$I_t = \nabla^\perp I \cdot \nabla \Delta I \tag{1}$$

for the image intensity I, where $\nabla^\perp$ denotes the perpendicular gradient $(-\partial_y, \partial_x)$ and Δ denotes the Laplace operator $\partial_x^2 + \partial_y^2$. Additional anisotropic diffusion of the image can produce a PDE of the form

$$I_t = \nabla^\perp I \cdot \nabla \Delta I + \nu \nabla \cdot (g(|\nabla I|) \nabla I). \tag{2}$$

The goal is to evolve (1) or perhaps more appropriately (2) to a steady state solution, satisfying the condition in the inpainting region that the isophote lines, in the direction of $\nabla^\perp I$, must be parallel to the level curves of the smoothness ΔI of the image intensity, which for $\nu = 0$ becomes

$$\nabla^\perp I \cdot \nabla \Delta I = 0. \tag{3}$$

We note that while the authors of [2] present their method as moving image intensity along isophote lines, there is another sense in which to think of the dynamics. Equation (1) is a transport equation that convects the image intensity I along level curves of the smoothness, ΔI. This can be seen by noting that (1) is equivalent to $DI/Dt = 0$ where D/Dt is the material derivative $\partial/\partial t + v \cdot \nabla$ for the velocity field $v = \nabla^\perp \Delta I$. In particular I is convected by the velocity field v which is in the direction of level curves of the smoothness ΔI. In the next section we discuss how

this is related to a classical problem in incompressible fluid dynamics. Our goal is to make use of expertise developed in that field to design better inpainting algorithms.

2. Analogy to Transport of Vorticity in Incompressible Fluids

Incompressible Newtonian fluids are governed by the Navier-Stokes equations, which couple the velocity vector field $\vec{v}$ to a scalar pressure p:

$$v_t + v \cdot \nabla v = -\nabla p + \nu \Delta v, \quad \nabla \cdot v = 0. \tag{4}$$

In two space dimensions, the divergence free velocity field v possesses a stream function Ψ satisfying $\nabla^\perp \Psi = v$. In addition, in 2D the vorticity, $\omega = \nabla \times v$, satisfies a very simple advection diffusion equation, which can be computed by taking the curl of the first equation in (4) and using some basic facts about the geometry in 2D:

$$\omega_t + v \cdot \nabla \omega = \nu \Delta \omega. \tag{5}$$

Note here that in 2D the vorticity is a scalar quantity that is related to the stream function through the Laplace operator, $\Delta \Psi = \omega$. In the absence of viscosity $\nu = 0$, we obtain the Euler equations for inviscid flow. Both the inviscid and viscous problems, with appropriate boundary conditions, are globally well-posed in two space dimensions. Solutions exist for any smooth initial condition and they depend continuously on the initial and boundary data [17, 18, 23, 24, 36, 40].

In terms of the stream function, equation (5) implies that steady state inviscid flows must satisfy

$$\nabla^\perp \Psi \cdot \nabla \Delta \Psi = 0 \tag{6}$$

which says that the Laplacian of the stream function, and hence the vorticity, must have the same level curves as the stream function. The analogy to image inpainting in the previous section is now clear: the stream function for inviscid fluids in 2D satisfies the same equation as the steady state image intensity equation (3).

The point is that in order to solve the inpainting problem proposed in the previous section, we have to find a steady state stream function for the inviscid fluid equations, which is a problem possessing a rich and well developed history.

2.1. The stream function-image intensity analogy

The main analogy that we build on in this paper is the parallel between the stream function in a 2D incompressible fluid and the role of image intensity function I in the inpainting method described in Section 1.2. This allows us to design a new inpainting method that will achieve the same steady equation (3).

Let Ω be a region in the plane in which we want to inpaint from surrounding data. Assume that the image intensity I_0 is a smooth function (with possibly large gradients) outside of Ω and we know both I_0 and ΔI_0 on the boundary $\partial\Omega$. We now design a 'Navier-Stokes' based method for image inpainting. In this method the fluid dynamic quantities have the following parallel to quantities in the inpainting method.

Navier-Stokes	Image inpainting
stream function Ψ	Image intensity I
fluid velocity $v = \nabla^\perp \Psi$	isophote direction $\nabla^\perp I$
vorticity $\omega = \Delta \Psi$	smoothness $w = \Delta I$
fluid viscosity ν	anisotropic diffusion ν

Our goal is to solve a form of the Navier-Stokes equations in the region to be inpainted. The method described next is based on the vorticity-stream form (5) of the Navier-Stokes equations, however it is also possible to consider other methods based on the primitive variables form (4).

For ease of notation we denote by w the smoothness ΔI of the image intensity. Instead of solving a transport equation for I as in (2), we solve a vorticity transport equation for w:

$$\partial w / \partial t + v \cdot \nabla w = \nu \nabla \cdot (g(|\nabla w|) \nabla w), \tag{7}$$

where the function g allows for anisotropic diffusion of the smoothness w. The image intensity I which defines the velocity field $v = \nabla^\perp I$ in (7) is recovered by solving simultaneously the Poisson problem

$$\Delta I = w, \quad I|_{\partial\Omega} = I_0. \tag{8}$$

For $g = 1$, the direct numerical solution of of (7-8) is a classical way to solve both the dynamic fluid equations and to evolve the dynamics towards a steady state solution [30].

For fluid problems with small viscosity ν, the above dynamics can take a long time to converge to steady state, making the method less practical. Instead there are *pseudo-steady* methods that involve replacing the Poisson equation (8) with a dynamic relaxation equation

$$I_t - \alpha[\Delta I + w] = 0, \quad \alpha > 0, \quad I_{\partial\Omega} = I_0. \tag{9}$$

where the parameter α determines a rate of relaxation. In our situation, diffusion can result in a blurring of sharp interfaces, gradients of I, in the inpainting region. Hence it is often desirable to include anisotropic diffusion in the solution of I. This can be added directly to the dynamical problem (9) or as an additional step in conjunction with the Poisson step (8). In our test cases with anisotropic diffusion, we did not find a significant difference between direct solution of the Poisson equation in (8) versus the relaxation method (9).

Once steady state is achieved, we have effectively found a solution of (3) for the intensity I (perhaps modified slightly by the anisotropic diffusion). Hence we expect and indeed find that the Navier-Stokes inpainting method performs in many ways like the method proposed in [2]. There are several advantages to using the Navier-Stokes method. First, there is a well-developed theoretical and numerical literature for this problem on which we can build inpainting algorithms. Second there is the possibility to test the performance of the method against a number of classical examples from fluid dynamics. Finally, we are able to implement this method efficiently and we have a theoretical framework in which to understand the transport of information from the exterior into the inpainting region.

2.2. Isophote continuity and boundary conditions for Navier-Stokes

When using any PDE-based method to do inpainting, the issue of boundary conditions becomes very important. In order to produce a result which, to the eye, does not distinguish where the inpainting has taken place, we must at the very least continue both the image intensity and direction of the isophote lines continuously into the inpainting region. This means that any PDE-based method involving the image intensity I must enforce Dirichlet (fixed I) boundary conditions as well as a condition on the direction of ∇I on the boundary. Immediately we see that this poses a problem for lower-order PDE-based methods. Indeed, any first or second order PDE (including anisotropic diffusion) for the scalar I could typically only enforce one of these boundary conditions, the result being an inpainting with discontinuities in the slope of the isophote lines, or a method with a jump in I itself on the boundary. From a mathematical point of view, to fix this, one can either choose a higher order equation for I, as in [2], that requires more boundary conditions, or consider a vector evolution for ∇I, which is the idea of the Navier-Stokes method.

The Navier-Stokes analogy guarantees, in a very natural way, continuity of the image intensity function I and its isophote directions across the boundary of the inpainting region. First consider a solution of the Navier-Stokes equation (4) in primitive variables form satisfying the classical no-slip condition $v = 0$ on the boundary $\partial\Omega$. This condition guarantees two features: (a) that the stream function Ψ must be constant on the boundary, since the boundary is trivially a streamline of the flow; (b) that the direction of the fluid velocity $\vec{v}$ is always tangent to the boundary. A more general form of the no-slip boundary condition, for which well-posedness is known, is to prescribe the velocity vector $v = \vec{v}_0$ on the boundary. This would be the natural choice for a moving boundary. Specifying the velocity on the boundary is equivalent to specifying both the normal and tangential derivatives of the stream function Ψ on the boundary, since $v = \nabla^\perp \Psi$. However, specifying the tangential derivative of Ψ determines Ψ on the boundary up to a constant of integration, by simply integrating around the boundary with respect to its arc length. Similarly this information determines the direction of flow on the boundary. The result is that if we solve the Navier-Stokes equations with v fixed on the boundary, we obtain a solution with a stream function Ψ and velocity field v both of which are continuous up to the boundary.

For the Navier-Stokes inpainting method, we inherit the continuity across the boundary. For example, suppose we fix $\nabla^\perp I$ on the boundary. Then solving the Navier-Stokes inpainting equation with these boundary conditions will not only result in continuous isophotes, but also will produce an image intensity function that is continuous across $\partial\Omega$.

In this paper, we are interested in solving the vorticity stream form (7) which, practically speaking, requires boundary conditions for the stream function and the vorticity. For viscous Navier-Stokes, there are several well-studied methods for doing this, all of which involve some sort of Dirichlet condition for ω on the boundary. In the case of a fluid, information about ω outside of the boundary is typically not known, hence first and second order accurate methods use information about the stream function at the previous timestep in order to numerically compute $\Delta\Psi$ on the boundary. In the case of images, we have more information about ΔI outside the boundary and can use this to construct accurate boundary conditions for $w = \Delta I$ at the boundary. For a discussion of how this is treated in fluids, see [30] Chapter 6.

2.3. Existence and uniqueness of solutions

The Navier-Stokes based inpainting method inherits a theoretical framework from the mathematical theory of the Navier-Stokes equations. Although a full treatment of such issues is beyond the scope of this paper, we discuss the problem of uniqueness and its relevance to inpainting.

First we note that without the presence of viscosity in the method we do not have a unique *steady-state* solution. This can be seen in the following simple example, motivated by problems involving the mathematical concentration of vorticity in solution sequences of the Euler equations [8, 12]. Consider the problem (6) with $v = 0$ on $\partial\Omega$. Consider a disk D_{2R} of radius $2R$, centered at the point x_0, contained inside Ω. Let $w_1(r)$ and $w_2(r)$ be two *different* compactly supported functions on $[0, 1]$ so that $\int_0^1 w_i(r) r dr = 0$. From each w_i we can construct an exact "radial eddy" solution of the inviscid ($\nu = 0$) fluid equations: the vorticity ω_i is given as $\omega_i(x) = w_i(\frac{|x_0 - x|}{R})$, $|x - x_0| < R$, and zero otherwise. This results in a stream function satisfying (where

we denote $r = |x - x_0|$)

$$\Psi_i'(r) = \frac{1}{r}\int_0^r s\omega_i(s)ds$$

for $0 < r < 2R$. Note that $\Psi_i'(r) = 0$ for $R < r < 2R$. We can integrate the equation for Ψ_i' to recover Ψ_i where the constant of integration is chosen so that Ψ_i is identically zero in $R < r < 2R$. Outside of the disk D_{2R} we continue Ψ_i and ω_i as zero. Note that this is a C^∞ continuation, by construction. Since the initial functions w_1 and w_2 were chosen to be different, but both satisfying the mean zero constraint, the result is two different smooth solutions of the inviscid steady state problem. In terms of the actual image, the result would be two different bull's eye patterns on a disk in the interior of Ω. Note that this construction works for any disk or for multiple disks. So it is possible to construct a wide range of different solutions satisfying the same zero velocity boundary data.

At the other extreme when viscosity is sufficiently large, for smooth boundaries there is a unique solution of the steady Navier-Stokes equations [11]. For viscous flows with moderate and small viscosity the problem is more complex; for example, there are classical experimental examples in which known steady state solutions have varying stability properties depending on the viscosity. For example, in the case of Couette flow, the fluid is constrained to move between two concentric cylinders rotating at different speeds. There is an exact, radially symmetric, solution that satisfies the Navier-Stokes equation for all viscosities. However this solution is unstable for sufficiently high viscosity, resulting in the creation of multiple eddies in such experiments (see e.g. [37] p. 80-81).

We expect that Navier-Stokes based inpainting may inherit some of the stability and uniqueness issues known for incompressible fluids, although the effect of anisotropic diffusion is not clear. In fact, some degree of non-uniqueness of the steady state problem could be favorable for inpainting, since large inpainting regions might have multiple 'possible' solutions, the best choice of which might be determined by pattern matching or information from a previous frame, as in the case of video. Ultimately we hope that the choice of magnitude and anisotropy of viscosity, as well as the initial condition in the inpainting region, may determine the best possible outcome for the inpainting method.

Existence and uniqueness of time-dependent solutions of the Navier-Stokes equations with isotropic viscosity ($g = 1$ in (7)) is well established [17, 18, 23, 24, 36, 40]. In our problem we consider the dynamics with anisotropic viscosity, both at the level of the smoothness and at the level of the intensity. This method of selective smoothing is known to be ill-posed for a wide class of functions g, however the ill-posedness can be easily removed with a small amount of smoothness applied to the gradient inside the function g. Catte et. al. [6] established an existence and uniqueness theory which could be extended to include nonlinear transport terms from the Navier-Stokes equations. In fact, nonlinear diffusion has a natural physical analogy; Non-Newtonian fluids have a viscosity that depends locally and nonlinearly on the shear ($|\nabla v|$) of the fluid [32]. Another form of viscosity for Navier-Stokes include fourth order hyperviscosity which is commonly used in numerical simulations of turbulence [4].

3. Computational Examples

We now show via some examples how the Navier-Stokes inpainting method performs. In each example described below, the Navier-Stokes equations are solved in the inpainting region. We start by computing the vorticity w from the image I, using information from the exterior to determine the boundary vorticity. We evolve the vorticity stream form (7), using a simple forward Euler time stepping, with centered differences in space for the diffusion and a minmod method [29] for the convection term. The diffusion is anisotropic.

After one time step of (7) we compute the image intensity I by solving the Poisson equation (8) using the Jacobi iteration method. From this updated I we recompute w and start again. Every few steps we perform anisotropic diffusion on I, which helps to sharpen edges. Steady state is achieved after N iterations of this cycle, typically N=300. We can set the algorithm to stop automatically when I does not change appreciably. Parameters for the algorithm have been chosen in such a way as to work for a wide range of examples: $dt = 0.01$, $dx = dy = 1$, $\nu = 2$, 50 steps for Jacobi, 5 steps of anisotropic diffusion of I every 10 cycles. The evolution (7) might be made more efficient by using an ADI (alternate direction implicit) method [39].

When working with a color image, we perform inpainting on its three components separately (one luminance image and two chroma images), and join the results at the end (in the same was as in [2]). The algorithm is programmed in tens of lines of C++ code. The results shown here were obtained in a few seconds of CPU time of a standard PC under Linux.

3.1. Inpainting of stills

In this example we consider a still image, shown in Figure 1 top, with thick lines obscuring parts of the photograph. The bottom image in the figure shows the result of Navier-Stokes inpainting applied to the photo with the inpainting region corresponding to the obscuring lines.

Notice the result is sharp, edges are not blurred inside Ω nor do color artifacts appear. This is a clear example for which texture synthesis or manual selection algorithms are not a good choice, since the inpainting region is complex.

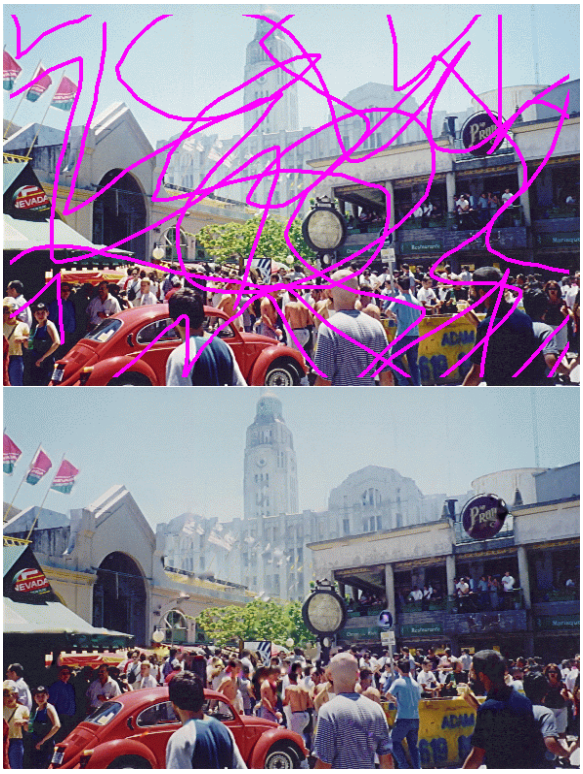


Figure 1: (top) Color photograph with lines obscuring parts of the image, (bottom) Navier-Stokes inpainting restoration of the photograph.

The topology of the inpainting region does not pose a problem, unlike the approach in [25, 26].

3.2. Video inpainting

Figure 2 shows four frames of the 'Foreman' video in which lettering overlay has been removed using Navier-Stokes inpainting. The inpainting is done frame-by-frame using the same method outlined in the previous subsection. Notice that this approach, though straightforward, shows very good results: sharp, no color artifacts, no motion artifacts.

3.3. Super-resolution

This example uses Navier-Stokes inpainting to increase the resolution of an image. We have taken a detail I of size (n,m) of the original image, the right eye of the girl in Figure 3. Then we increased its size 9-fold with replication (zeroth order interpolation) to obtain I' of size (9n,9m). Finally we apply Navier-Stokes inpainting to I', while fixing I and $\nabla^{\perp} I$, from the original small image, on the lattice of points with coordinates (9i,9j). Notice how inpainting reproduces a round iris.

In this example, we use Navier-Stokes inpainting only for the brightness component of the color image. A simpler filter is used for the chroma components. Better results may be achieved via harmonic map smoothing of vectors (see [35]). For an axiomatic/PDE based technique for image iterpolation, see [5].

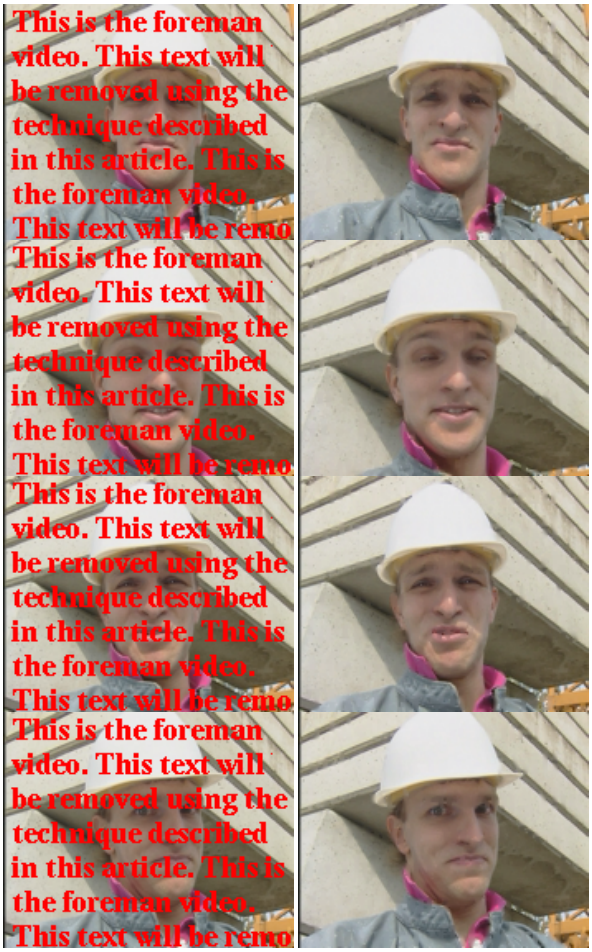


Figure 2: Four frames from the 'Foreman' video in which the lettering has been removed using Navier-Stokes inpainting (available electronically).

4. Summary and Conclusions

In this paper, we show the importance of computational fluid dynamics (CFD) in general, and Navier-Stokes equations in particular, to vision problems. Although CFD ideas have been used in computer graphics for such problems as modeling natural phenomena, in shape analysis following [29], and as an interpretation for anisotropic diffusion [27], the connection and application here is to the best of our knowledge novel. Having such a direct interdisciplinary connection has several benefits for the computer vision community. First it brings well-established numerical methods and theory to a problem of central importance in

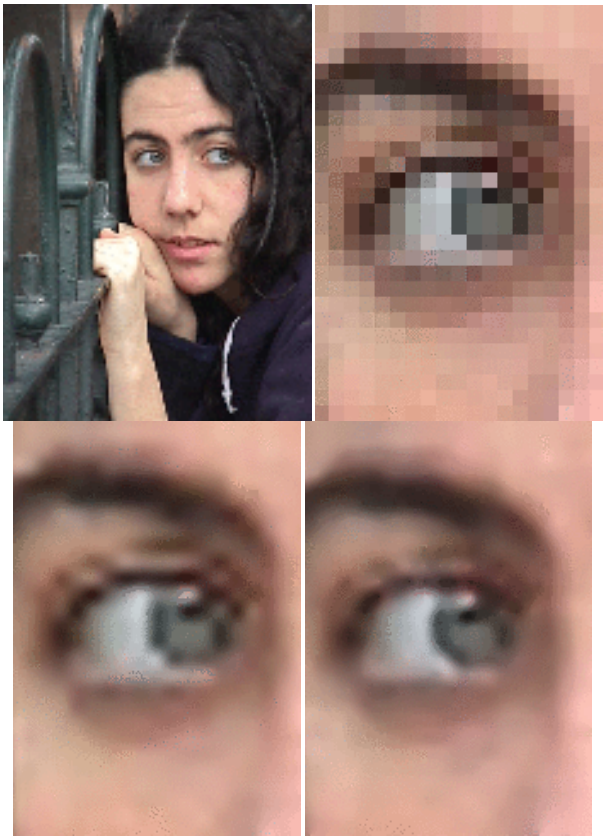


Figure 3: Top left: Original image of girl, 120X160 pixels. Top right: one of her eyes (19x27 pixels) magnified with zeroth order zoom to 169x241 pixels, Bottom left: bicubic interpolation, Bottom right: Navier-Stokes inpainting. Note how the edges are smoother with the NS approach, e.g., the eye ball.

vision research. Second, the very close connection to CFD has the potential to draw more people from that community to explore problems in image processing and computer vision. This can only serve to accelerate the development of the field.

Although we present a new method that seems to perform well, there are several interesting problems and issues to be worked out that should give improvements. First there is the design of the numerical algorithm. We use a vorticity-stream based method due to its direct connection to the method introduced in [2]. However one can consider many other methods, including direct solution of the primitive variables form of the Navier-Stokes equation (4) with linear viscosity replaced by anisotropic diffusion. There is a vast literature of CFD methods that can be modified to suit the image problems. Moreover, the close analogy between Navier-Stokes and the method described in [2] suggests the possibility to try hybrid methods that cross between the two equations, thereby giving a range of parameters that can be tuned to optimize the end result. Moreover, the theoretical knowledge base developed in connection with Navier-Stokes might provide more insight into a theory for the PDE proposed in [2]. Finally there is a very interesting connection that we have not yet explored related to the dynamical evolution of the fluid equations and stability of solutions. For example it is possible that classical two-dimensional fluid dynamics instabilities, like the Kelvin-Helmholtz instability [19, 14] for shear layers, might allow us to generate interesting pattern formation in large inpainting regions. The control of such instabilities should be linked to the magnitude of diffusion present in the model and to the choice of initial condition.

At the very practical level, we have not dealt with textures, and the parameters are set manually. In this respect we believe that the user may be required to tune both the parameters and initial data in the inpainting region to suit the particular problem at hand. However, our experience suggest that large classes of problems can be efficiently solved with the same or similar ranges of parameters and initial conditions.

We also expect that this connection, when extended to higher dimensions, will allow us to generalize the inpainting problem to other features (e.g., optical flow) and other modalities. We are currently working on a video inpainting technique, based on the framework presented here, to automatically switch between texture and structure inpainting, as well as to permit the use of information from other frames when it is available.

Acknowledgments

We thank S. Betelú, P. Constantin, R. Kohn, and A. Pardo for helpful comments. This work was supported by IIE Uruguay, ONR grants N00014-01-1-0290 and N00014-97-1-0509, the ONR Young Investigator Award, the Presidential Early Career Awards for Scientists and Engineers (PECASE), a NSF CAREER Award, and by the NSF Learning and Intelligent Systems Program (LIS).

References

[1] C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera, "Filling-In by Joint Interpolation of Vector Fields and Gray Levels," *IEEE Trans. Img. Proc.*, to appear.

[2] M. Bertalmio, G. Sapiro, C. Ballester and V. Caselles, "Image inpainting," *Computer Graphics, SIGGRAPH 2000*, pp. 417-424, July 2000.

[3] C. Braverman. *Photoshop retouching handbook*. IDG Books Worldwide, 1998.

[4] G. L. Browning and H. O. Kreiss. "Comparison of numerical methods for the calculation of two-dimensional turbulence", *Math. Comp.*, **52(186)**, 369-388, 1989.

[5] V. Caselles, J. M. Morel, and C. Sbert, "An axiomatic approach to image interpolation", *IEEE Trans. Img. Proc.*, March 1998.

[6] F. Catte, P. Lions, J. Morel, and T. Coll, "Image selective smoothing and edge detection by nonlinear diffusion" *SIAM J. Num. Anal.*, **29**, 1: 182-193, 1992.

[7] T. Chan and J. Shen, "Mathematical models for local deterministic inpaintings," *UCLA CAM Report* **00-11**, March 2000 (available at www.math.ucla.edu).

[8] R. J. DiPerna and A. J. Majda, "Concentrations in Regularizations for 2-D Incompressible Flow", *Commun. Pur. Appl. Math.*, **40**, 301-345, 1987.

[9] A. Efros and T. Leung, "Texture synthesis by non-parametric sampling," *Proc. IEEE International Conference Computer Vision*, pp. 1033-1038, Corfu, Greece, September 1999.

[10] G. Emile-Male, *The Restorer's Handbook of Easel Painting*. Van Nostrand Reinhold, New York, 1976.

[11] G. P. Galdi, *An Introduction to the Mathematical Theory of the Navier-Stokes Equations, Vol II., Nonlinear Steady Problems*, Springer Tracts in Natural Philosophy, vol. 39, Springer-Verlag, New York, 1994.

[12] C. Greengard and E. Thomann, "On DiPerna-Majda concentration sets for two-dimensional incompressible flow", *Commun. Pur. Appl. Math.*, **41**, 295-303, 1988.

[13] D. Heeger and J. Bergen. *Pyramid based texture analysis/synthesis.* Computer Graphics, 229-238, SIGGRAPH 95, 1995.

[14] H. von Helmholtz. "Uber discontinuierliche flussigkeitsbewegungen", *Monatsber. Berlin. Akad.*, 215–228, 1868.

[15] A. Hirani and T. Totsuka. *Combining Frequency and spatial domain information for fast interactive image noise removal*. Computer Graphics, pp. 269-276, SIGGRAPH 96, 1996.

[16] L. Joyeux, O. Buisson, B. Besserer, S. Boukir. "Detection and removal of line scratches in motion picture films," *Proceedings of CVPR'99, IEEE Int. Conf. on Computer Vision and Pattern Recognition*, Fort Collins, Colorado, USA, June 1999.

[17] V. Judovich, *Math. Sb. N. S.*, **64**, 562-588, 1964.

[18] T. Kato, "On classical solutions of the two dimensional nonstationary Euler equation", *Arch. Rat. Mech. Anal.*, **25**, 188-200, 1967.

[19] Lord Kelvin. *Nature*, 50:524, 549, 573, 1894.

[20] D. King. *The Commissar Vanishes*. Henry Holt and Company, 1997.

[21] A.C. Kokaram, R.D. Morris, W.J. Fitzgerald, P.J.W. Rayner, "Detection of missing data in image sequences," *IEEE Trans. Img. Proc.* **11(4)**, 1496-1508, 1995.

[22] A.C. Kokaram, R.D. Morris, W.J. Fitzgerald, P.J.W. Rayner. "Interpolation of missing data in image sequences," *IEEE Trans. on Img. Proc.* **11(4)**, pp. 1509-1519, 1995.

[23] O. A. Ladyzhenskaya, *The Mathematical Theory of Viscous Incompressible Flow*, Gordon and Breach Science Publishers, New York-London, 1963.

[24] A. J. Majda and A. L. Bertozzi, *Vorticity and incompressible flow*, Cambridge Univ. Press, 2001.

[25] S. Masnou, *Filtrage et Desocclusion d'Images par Méthodes d'Ensembles de Niveau*, Thèse, Univ. Paris-Dauphine, 1998.

[26] S. Masnou and J.M. Morel, *Level-lines based disocclusion*. 5th IEEE Int'l Conf. on Image Processing, Chicago, IL. Oct 4-7, 1998.

[27] J. Monteil and A. Beghdadi, "A new interpretation and improvement of the Nonlinear Anisotropic Diffusion for Image Enhancement", *IEEE Trans. Patt. Anal. Mach. Int.*, **21(9)**, 940-946, 1999.

[28] M. Nitzberg, D. Mumford, and T. Shiota, *Filtering, Segmentation, and Depth*, Springer-Verlag, Berlin, 1993.

[29] S. Osher and J. Sethian, "Fronts propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulations," *J. Comp. Phys.*, **79**, 12-49, 1988.

[30] R. Peyret and T. D. Taylor, *Computational methods for fluid flow*, Springer Verlag, 1983.

[31] D. L. Ringach and R. Shapley, "Spatial and temporal properties of illusory contours and amodal completion", *Vision Research* **36**, 3037-3050, 1996.

[32] M. Renardy, *Mathematical Analysis of Viscoelastic Flows*, CBMS-NSF Conference Series in Applied Mathematics 73, Soc. Ind. Appl. Math., Philadelphia, 2000.

[33] L. Rudin, S. Osher and E. Fatemi. "Nonlinear total variation based noise removal algorithms," *Phys. D* **60**, 259-268, 1992.

[34] E. Simoncelli and J. Portilla, *Texture characterization via joint statistics of wavelet coefficient magnitudes*. 5th IEEE Int'l Conf. on Image Processing, Chicago, IL. Oct 4-7, 1998.

[35] B. Tang, G. Sapiro, and V. Caselles, "Diffusion of general data on non-flat manifolds via harmonic maps theory: The direction diffusion case," *Int. Journal Computer Vision* **36:2**, pp. 149-161, February 2000.

[36] R. Temam, *Navier-Stokes Equations*, North–Holland Publishing Company, Amsterdam, second edition, 1986.

[37] M. VanDyke, *An Album of Fluid Motion*, The Parabolic Press, Stanford, CA, 1983.

[38] S. Walden, *The Ravished Image*. St. Martin's Press, New York, 1985.

[39] J. Weickert, B. M. ter Haar Romeny, and M. A. Viergever, "Efficient and reliable schemes for nonlinear diffusion filtering", *IEEE Trans. Img. Proc.*, 7(3):398–410, March 1998.

[40] W. Wolibner, *Math. Zeit.*, **37**, 698-726, 1933.

Filling-In by Joint Interpolation of Vector Fields and Gray Levels

C. Ballester, * **M. Bertalmio**, † **V. Caselles**,* **G. Sapiro**,† and **J. Verdera**,*

April 8, 2000

Abstract

A variational approach for filling-in regions of missing data in digital images is introduced in this paper. The approach is based on joint interpolation of the image gray-levels and gradient/isophotes directions, smoothly extending in an automatic fashion the isophote lines into the holes of missing data. This interpolation is computed solving the variational problem via its gradient descent flow, which leads to a set of coupled second order partial differential equations, one for the gray-levels and one for the gradient orientations. The process underlying this approach can be considered as an interpretation of the Gestaltist's principle of good continuation. No limitations are imposed on the topology of the holes, and all regions of missing data can be simultaneously processed, even if they are surrounded by completely different structures. Applications of this technique include the restoration of old photographs and removal of superimposed text like dates, subtitles, or publicity. Examples of these applications are given. We conclude the paper with a number of theoretical results on the proposed variational approach and its corresponding gradient descent flow.

Keywords: Interpolation, filling-in, image gradients, image gray-levels, variational approach, partial differential equations, Gestalt principles.

1 Introduction

Filling-in missing data in digital images has a number of fundamental applications. They range from removing objects from a scene all the way to re-touching damaged paintings and photographs. The basic idea is to fill-in the gap of missing data in a form that it is non-detectable by an ordinary observer. In art, this process is called *inpainting* [38, 16, 25, 6, 7].

Since the early days of art and photography, filling-in and inpainting has been done by professional artists. Imitating their performance with semi-automatic digital techniques is currently an active area of research, see for example [23, 27, 28, 8] and the papers discussed below. The goal of this work is to introduce a novel algorithm for automatically filling-in gaps in the image. In this article we follow the suggestions in the conclusions section in [6] and introduce an energy

*Dept. de Tecnologia, University of Pompeu-Fabra, La Rambla 30-32, 08002 Barcelona, Spain, {coloma.ballester,vicent.caselles,joan.verdera}@tecn.upf.es

†Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA, {marcelo,guille}@ece.umn.edu

functional based on an interpretation of Gestaltist's good continuation principle. Suppose that we are given an image $u_0 : R \to [a, b]$, where R is a square of $\mathbb{R}^2$ and $a < b$, and Ω is an open bounded subset of R with Lipschitz continuous boundary. We shall call Ω the *hole* or *gap*. We want to fill-in the hole Ω based on the geometric and photometric information outside the hole. For that we use what we call a band around Ω, i.e., we consider an open region $\tilde{\Omega}$ of R such that $\overline{\Omega} \subseteq \tilde{\Omega}$. The band we refer to will be the set $B = \tilde{\Omega} \setminus \overline{\Omega}$. To fill-in the hole Ω we use the information of u_0 contained in B, mainly gray level and the vector field of isophotes (level sets) directions of u_0 in B. We attempt to continue the level sets of u in B inside Ω taking into account the principle of good continuation. We propose an energy functional which takes into account these principles interpreted in a suitable way. The energy functional we propose has to be minimized with respect to two variables : a vector field θ which represents the directions of the level lines of u, and the gray level u. θ and u are constrained in the band B by their known values there. The use of the vector field of directions θ is one of the main points of the algorithm presented in this paper, which permits the level sets to smoothly continue inside the hole. We are then continuing both the geometric and photometric properties of the image inside the hole.

Let us finally say that the only user interaction required by the algorithm here introduced is to mark the regions to be filled-in. Although a number of techniques exist for the semi-automatic detection of image defects (mainly in films), addressing this is not part of the scope of this paper. Since the algorithm here presented can be used not just to restore damaged photographs but also to remove undesired objects and writings on the image, the regions must be marked by the user, since they depend on his/her subjective selection.

1.1 Closely related approaches

Before proceeding with the detailed description of our algorithm, let us comment on related work. Note that image denoising is different to filling-in, since the regions of missing data are usually large. That is, regions occupied by top to bottom scratches along several film frames, long cracks in photographs, superimposed large fonts, and so on, are of significant larger size than the type of noise assumed in common image enhancement algorithms. In addition, in common image enhancement applications, the pixels contain both information about the real data and the noise (e.g., image plus noise for additive noise), while in our application there is no significant information in the region to be inpainted.

A very active area related to the work here presented is the restoration of damaged films. The basic idea here is to use information from past and future frames to restore the current one, e.g., [23, 28]. Of course, this general approach can not be used when dealing with still images.

Another area related to the work here described is texture synthesis. The basic idea here is to select a texture and synthesize it inside the region to be filled-in (the hole). Although outstanding texture synthesis results have been reported in the literature, e.g., [22, 14, 19, 35], these algorithms require the user to select the texture to be copied into the hole. For images where the region to be replaced covers several different structures, the user would need to go through the tremendous work of segmenting them and searching corresponding replacements throughout the picture. Although part of this search can be done automatically, this is extremely

time consuming and requires the non-trivial selection of many critical parameters, e.g., [14].

Last, a number of fundamental works on disocclusion and line continuation have been reported in the literature, and these are the closest to our approach. A pioneering contribution in this area is described in [33]. The authors presented a technique for removing occlusions with the goal of image segmentation. Since the region to be filled-in can be considered as occluding objects, removing occlusions is analogous to image inpainting. The basic idea suggested by the authors is to connect T-junctions at the same gray-level with elastica minimizing curves (see later in this paper for the exact definition of elastica curves). The technique was mainly developed for simple images, with only a few objects with constant gray-levels, and will not be applicable for the examples with natural images presented later in this paper. (Other researchers, e.g., D. Jacobs, R. Basri, S. Zucker, etc, have followed this interesting research area, mainly developing techniques for smooth curve continuation.) Masnou and Morel [31, 32] recently extended these ideas, presenting a very elegant and inspiring general formal variational formulation for disocclusion and a particular practical algorithm implementing some of the ideas in this formulation. The algorithm fills-in by joining with geodesic curves the points of the isophotes arriving at the boundary of the region to be inpainted. The holes in their algorithm are limited to having simple topology. In addition, the angle with which the level lines arrive at the boundary of the holes are not (well) preserved, and the algorithm uses straight lines to join equal gray value pixels. On the other hand, we should note that this work has motivated in part and inspired our work.

Recently, we have addressed the concept of smooth continuation of information in the level-lines direction in [6, 7]. We proposed an algorithm, inspired in partial differential equations, that propagates the image Laplacian in this direction. The algorithm attempts to imitate basic approaches used by professional restorators. The algorithm also introduced the importance of propagating both the gradient direction (geometry) and gray-values (photometry) of the image in a band surrounding the hole to be filled-in. It is part of the goal of the current paper to adopt some of the ideas of [6, 7], while deviating from the particular model in order to be able to define a formal variational approach to the filling-in problem.

The work in [6, 7] inspired a very elegant approach to the filling-in problem recently reported in [9] (this work was performed independently to the one reported in this paper).[1] The authors present a clear and very intuitive axiomatic approach to the problem. The main algorithm they propose after an interesting discussion of the inpainting problem is to minimize the Total Variation (TV), [34], of the image inside the hole (they also use, as proposed in [6, 7] and further studied in this paper, a band surrounding the region). They address in addition the interpolation and filling-in in the presence of noise, a very important additional contribution. As in the work of Masnou and Morel, their interpolation is limited to creating straight isophotes, not necessarily smoothly continued from the hole boundary, and mainly is developed (as the authors clearly state) for small holes. Although straight connections give visually pleasant results for small holes, it is important to develop a theory that permits interpolation of level lines across large gaps. As we will argue later in this paper, in order to obtain such a smooth interpolation of isophotes, it is necessary to go into high order PDE's or systems of PDE's, as done in [6, 7] and

[1] We thank the authors for providing us with a preliminary report of their work.

here.

To conclude this section, the interested reader is referred to the works of Nitzberg-Mumford-Shiota, Masnou-Morel, and Chan-Shen (as well as our previous work) to study other interesting and very related techniques for filling-in.

Let us conclude with the plan of the paper. In Section 2 we introduce the problem, the functional spaces and the energy functional for image inpainting. To clarify the meaning of the functional, we also discuss the particular case where we interpolate the gray level, knowing the vector field of directions. Section 3 is devoted to numerical experiments. Section 4 contains some conclusions. Finally, the Appendix is devoted to the proof of existence of minimizers for the energy functional introduced in this paper.

2 Joint interpolation of vector fields and gray values

Let $u_0 : R \to I\!R$ be an image defined on a domain R of $I\!R^2$, which we may suppose to be a square. Let $\Omega, \tilde{\Omega}$ be two open bounded domains in $I\!R^2$ with Lipschitz boundary and suppose that $\overline{\Omega} \subseteq \tilde{\Omega} \subset\subset R$. To simplify our presentation we shall assume that $\tilde{\Omega}$ does not touch the boundary of the image domain R. Let $B := \tilde{\Omega} \setminus \overline{\Omega}$. B will be called the band around Ω. Suppose that a function u_0 is given in B which, for the moment being, we shall assume to be smooth in $\overline{B}$ (later we shall assume that u_0 is of bounded variation, i.e., $u_0 \in BV(B)$). Let θ_0 be the vector field of directions of the gradient of u_0 on B, i.e., θ_0 is a vector field with values in $I\!R^2$ satisfying $\theta_0(x) \cdot \nabla u_0(x) = |\nabla u_0(x)|$ and $|\theta_0(x)| \leq 1$ (ideally 1 or 0, see below). This is the information we shall use on B.

We pose the image inpainting problem in the following form: Can we extend (in a reasonable way) the pair of functions (u_0, θ_0) from the band $\tilde{\Omega} \setminus \overline{\Omega}$ to a pair of functions (u, θ) defined inside Ω ? Of course, we will have to precise what we mean by a reasonable way. We shall discuss and analyze a variational formulation of this filling-in problem and discuss possible energy functionals, and their corresponding gradient descent flows, which give a solution to it. The data are given on the band B and we should constraint the solution (u, θ) to be near the data on B. The vector field θ should satisfy $|\theta| \leq 1$ on Ω and should be related to u by trying to impose that $\theta \cdot \nabla u = |\nabla u|$, i.e., we should impose that θ is related to the vector field of directions of the gradient of u. The condition $|\theta(x)| \leq 1$ should be interpreted as a relaxation of this. Indeed, it may happen that $\theta(x) = 0$ (flat regions) and then we cannot normalize the vector field to a unit vector. We should have in mind that the ideal case would be that $\theta = \frac{\nabla u}{|\nabla u|}$, u being a smooth function with $\nabla u(x) \neq 0$ for all $x \in \Omega$. Finally, we should impose that the vector field θ_0 in the band tries to continue smoothly to θ inside Ω. We shall impose this by observing that, in case θ represents the directions of the normals to the level lines of u, i.e., of the curves $u(x_1, x_2) = \lambda$, $\lambda \in I\!R$, then a term like $div(\theta)$ represents its curvature. Motivated by the principle of smooth continuation, our energy functional should contain terms integrating $div(\theta)$. Indeed, collecting all the observations above, we propose to minimize a functional of the form

$$\text{Minimize} \int_{\tilde{\Omega}} |div(\theta)|^p (a + b|\nabla k * u|) dx + \alpha \int_{\tilde{\Omega}} (|\nabla u| - \theta \cdot \nabla u) dx, \tag{1}$$

where a, b, and α are positive constants and k is a smoothing kernel. We need to give a sense to the integrals appearing in the above expression and to make precise the admissible class of functions where the functional has to be minimized. For that, we need to introduce some function spaces. This is done in the following section. Once this has been formally addressed, we will further discuss the underlying concepts of the above functional.

2.1 Function spaces

Let us first recall the definition of BV functions and total variation. Let Q be an open set. A function $u \in L^1(Q)$ whose partial derivatives in the sense of distributions are measures with finite total variation in Q is called a function of bounded variation. The class of such functions will be denoted by $BV(Q)$. Thus $u \in BV(Q)$ if and only if there are Radon measures $\mu_1, \ldots, \mu_N$ defined in Q with finite total mass in Q and

$$\int_Q u D_i \varphi dx = -\int_Q \varphi d\mu_i \tag{2}$$

for all $\varphi \in C_0^\infty(Q)$, $i = 1, \ldots, N$. Thus the gradient of u is a vector valued measure with finite total variation

$$\| \nabla u \| = \sup\{\int_Q u \, div \, \varphi \, dx : \varphi \in C_0^\infty(Q, I\!R^n), |\varphi(x)| \leq 1 \text{ for } x \in Q\}. \tag{3}$$

The space $BV(Q)$ is endowed with the norm

$$\| u \|_{BV} = \| u \|_{L^1(Q)} + \| \nabla u \| . \tag{4}$$

We say that a measurable set $E \subseteq Q$ has *finite perimeter* in Q if its indicator function $\chi_E \in BV(Q)$. If $u \in BV(Q)$ almost all its level sets $[u \geq \lambda] = \{x \in Q : u(x) \geq \lambda\}$ are sets of finite perimeter. For sets of finite perimeter E one can define the essential boundary $\partial^* E$, which is rectifable with finite H^{N-1} measure, and compute the normal to the level set at H^{N-1} almost all points of $\partial^* E$. Thus at almost all points of almost all level sets of $u \in BV(Q)$ we may define a normal vector $\theta(x)$. This vector field of normals θ can be also defined (hence extended to all Q) as the Radon-Nikodym derivative of the measure ∇u with respect to $|\nabla u|$, i.e., it formally satisfies $\theta \cdot \nabla u = |\nabla u|$ and, also, $|\theta| \leq 1$ a.e.. For further information concerning functions of bounded variation we refer to [1, 17, 39].

Let us now introduce the function spaces for θ. Let Q be an open bounded subset of $I\!R^2$ with a Lipschitz boundary. We define

$$W^{1,p}(div, Q) = \{\theta \in L^p(Q)^2 : div(\theta) \in L^p(Q)\}, \quad 1 \leq p < \infty,$$

and

$$M(div, Q) = \{\theta \in L^1(Q)^2 : div(\theta) \quad \text{is a Radon measure in } Q\}.$$

The Trace Theorem ([2],[10]) guarantees that the normal component $\theta \cdot n|_{\partial Q}$, is well defined for vector fields θ in $W^{1,p}(div, Q)$, or in $M(div, Q)$. To simplify our notation we shall assume that $W^{1,1}(div, Q)$ represents the space $M(div, Q)$.

Next, we shall give a sense to the integrals of bounded vector fields with divergence in L^p integrated with respect to the gradient of a BV function. For that, we shall need some results from [2] (see also [26] and [11]). Let Q be an open bounded subset of $\mathbb{R}^n$ with Lipschitz continuous boundary. Let $p \geq 1$ and $q \geq 1$ be such that $\frac{1}{p} + \frac{1}{q} = 1$. Following [2], let

$$X(Q)_p = \{z \in L^\infty(Q, \mathbb{R}^n) : div(z) \in L^p(Q)\}. \tag{5}$$

If $z \in X(Q)_p$ and $w \in BV(Q) \cap L^q(Q)$ we define the functional $(z, \nabla w) : C_0^\infty(Q) \to \mathbb{R}$ by the formula

$$< (z, \nabla w), \varphi >= -\int_Q w\, \varphi\, div(z)\, dx - \int_Q w\, z \cdot \nabla \varphi\, dx. \tag{6}$$

Then $(z, \nabla w)$ is a Radon measure in Q,

$$\int_Q (z, \nabla w) = \int_Q z \cdot \nabla w\, dx \tag{7}$$

for all $w \in W^{1,1}(Q) \cap L^p(Q)$ and

$$\left| \int_B (z, \nabla w) \right| \leq \int_B |(z, \nabla w)| \leq \|z\|_\infty \int_B \|\nabla w\| \tag{8}$$

for any Borel set $B \subseteq Q$.

In [2], a weak trace on ∂Q of the normal component of $z \in X(Q)_p$ is defined. Concretely, it is proved that there exists a linear operator $\gamma : X(Q)_p \to L^\infty(\partial Q)$ such that

$$\|\gamma(z)\|_\infty \leq \|z\|_\infty$$

$$\gamma(z)(x) = z(x) \cdot \nu(x) \quad \text{for all } x \in \partial Q \text{ if } z \in C^1(\overline{Q}, \mathbb{R}^N).$$

We shall denote $\gamma(z)(x)$ by $[z, \nu](x)$. Moreover, the following *Green's formula*, relating the function $[z, \nu]$ and the measure $(z, \nabla w)$, for $z \in X(Q)_p$ and $w \in BV(Q) \cap L^q(Q)$, is established:

$$\int_Q w\ div(z)\ dx + \int_Q (z, \nabla w) = \int_{\partial Q} [z, \nu] w\ dH^{N-1}. \tag{9}$$

If no confusion arises, we shall denote $z \cdot \nabla w$ instead of $(z, \nabla u)$ for $z \in X(Q)_p$, $w \in BV(Q) \cap L^q(Q)$

2.2 The energy derivation and interpretation

One of the key concepts above was the band around the hole. The band is of local character but in principle it could be extended to all the known part of the image. Obviously, what happens at distant parts can be independent or not from what happens at the hole, but, in our construction below, we suppose that only a narrow band around the hole influences what happens inside the hole. Could we fill-in without the band ? To discuss this suppose that we are given the image of Figure 1.a which is a gray band on a black background partially occluded by a square Ω. We suppose that the sides of the square hole Ω are orthogonal to the level lines of the original image. In these conditions, the normal component of the vector field θ_0 outside Ω is null at $\partial\Omega$. Thus if the boundary data is just $\theta_0 \cdot n|_{\partial\Omega}$, we would have that $\theta_0 \cdot n|_{\partial\Omega} = 0$. In particular, the vector

field $\theta = 0$ satisfies this condition. If we are not able to propagate θ inside Ω this may become an unpleasant situation, since this would mean that we do no propagate the values of u at the boundary. If we write the functional (1) with $\theta = 0$, it turns out to be the Total Variation [34]. The decision of extending the gray band or filling the hole with the black level would be taken as a function of the perimeter of the discontinuities of the function in the hole. To overcome this situation we introduce the band which ensures that the vector field outside Ω is present in the functional. In Figure 1.b we display the result of the interpolation with $\theta = 0$ on Ω. In Figure 1.c we display the result of the interpolation using the functional we shall completely describe below, which takes into account the band B and computes the vector field θ in $\tilde{\Omega} = \overline{\Omega} \cup B$.

Figure 1.a. Figure 1.b. Figure 1.c.

Thus, let B be a band around Ω with a Lipschitz boundary containing the boundary of Ω (see Figure 2). As we made explicit above, $B = \tilde{\Omega} \setminus \overline{\Omega}$. Given the band B and the function u_0 of bounded variation in B, we define the space

$$BV(\tilde{\Omega}, B, u_0) = \{u \in BV(\tilde{\Omega}) : u = u_0 \text{ in } B\}.$$

Let $\theta_0 : B \to I\!R^2$ be a vector field of directions of the gradient of u_0, i.e., $|\theta_0| \leq 1$ and $\theta_0 \cdot \nabla u_0 = |\nabla u_0|$ a.e. in B. In practice we shall constraint the vector field θ to be the vector field of directions of u_0 only indirectly, through the functional. We could also introduce this as a constraint or with a penalty term $\int_B (\theta - \theta_0)^2$ (see also [9, 34] for penalties of this form in the gray values).

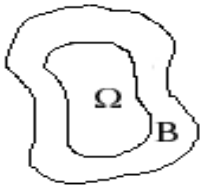


Figure 2.

Combining the previous elements, the band, the relations between θ and u, and the smoothness term on θ, we propose to interpolate the pair (θ, u) in Ω by minimizing the functional:

$$\begin{array}{l} \text{Minimize } \int_{\tilde{\Omega}} |div(\theta)|^p (a + b|\nabla k * u|) dx + \alpha \int_{\tilde{\Omega}} (|\nabla u| - \theta \cdot \nabla u) dx \\ \theta \in W^{1,p}(div, \tilde{\Omega}) \\ u \in BV(\tilde{\Omega}, B, u_0) \\ \theta \cdot n|_{\partial\tilde{\Omega}} = \theta_0 \cdot n|_{\partial\tilde{\Omega}} \\ |\theta| \leq 1 \\ |u| \leq \|u_0\|_{L^\infty(B)}. \end{array} \tag{10}$$

where $a, \alpha > 0$, $b \geq 0$ and k denotes a regularizing kernel of class C^1 such that $k(x) > 0$ a.e.. The previous functional is coercive and admits a minimum in the class of functions described above if $p > 1$. The case $p = 1$ is under study. The functional can be interpreted as a formulation of the principle of good continuation and amodal completion as formulated in the Gestalt theory of vision. The following remarks contain heuristic arguments which may help to understand our choice. In next subsection we shall explain in more detail the role of the term coupling θ and u.

Remarks.

1. The constant b is ≥ 0. If u is the characteristic function of the region enclosed by a curve C then a term like

$$\int_{\tilde{\Omega}} |div(\theta)|^p |\nabla u| \tag{11}$$

is related to $\int_C |\kappa|^p ds$, where κ is the Euclidean curvature (of the level-sets). If $p = 2$, this term appears in Euler's elastica,

$$\int_C (\alpha + \beta\kappa^2) ds, \quad \alpha, \beta > 0. \tag{12}$$

Euler's elastica (12) was proposed in [33] as a technique for removing occlusions with the goal of image segmentation, since this criterion yields smooth, short, and not too curvy curves. In terms of characteristic functions, Euler's elastica can be written as

$$\int |\nabla u| \left(\alpha + \beta \left| div\left(\frac{\nabla u}{|\nabla u|}\right)\right|\right)^2. \tag{13}$$

In [5], it was shown that this functional is not lower semicontinuous. The functional proposed by Masnou and Morel [31], [32] can be interpreted as a relaxation of it, since it integrates functionals like the elastica (plus the angle that the curve makes with the corresponding level line arriving at the boundary) along the level lines of the function u. Our functional can be also considered as a relaxed formulation of the energy of the elastica. For that, we introduced θ as a independent variable, and we tried to couple it to u with the term

$$\int_{\tilde{\Omega}} |\nabla u| - \theta \cdot \nabla u, \tag{14}$$

so that, heuristically, we try to impose that $\theta \cdot \nabla u = |\nabla u|$ (see next Subsection for a detailed discussion of this term). Finally, let us say that for mathematical reasons we have

convolved the ∇u term of (11) to be able to prove the existence of a minimum for (10). From a theoretical point of view, this may invalidate our previous comments. But, from a practical point of view, it gives a weight to the curve of discontinuities of the image.

2. The constant a has to be > 0. Otherwise we do not get compactness on θ. Now, let us comment on the two terms containing $div(\theta)$. Heuristically, if we do not compute θ in a proper way, in a continuous image like in Figure 1, θ could be zero except on a set of curves. Then $\theta = 0$ a.e. on B (or on $\tilde{\Omega}$) and a term like

$$\int_{\tilde{\Omega}} |div(\theta)|^p dx \tag{15}$$

would produce a null value since $div(\theta) = 0$. On the other hand, a term like (11) would integrate a power of the curvature on the level line corresponding to the discontinuity of the image and it would guarantee that the functional is not null. This argument is only heuristic and not completely justified. Indeed, we believe that in such example as in Figure 1, a term like (15) would induce a regularizing effect on θ and the support of θ would not be a curve any more. In that case, the integral (15) would not be null.

3. Related to the question discussed in the last comment is the possibility to compute a regularized vector field of directions for images which are constant except at jump discontinuities. A direct computation of the vector field outside the hole in an image like Figure 1.a gives a null vector field at all points except the points on the level line separating the black from the white region. This may be not be a good starting point to extend reasonably the vector field θ inside Ω. To initialize the algorithm of steepest descent, a regularization of θ outside Ω may be constructed as the vector field of directions of the image $U_0(t,x)$ obtained by regularizing $u_0(x)$, $x \in B$, with the equation

$$\begin{aligned} &\frac{\partial u}{\partial t} = div\left(\frac{\nabla u}{|\nabla u|}\right) && \text{in } Q = (0,\infty) \times B \\ &\frac{\partial u}{\partial \eta} = 0 && \text{in } S = (0,\infty) \times \partial B \\ &u(0,x) = u_0(x) && \text{for } x \in B. \end{aligned} \tag{16}$$

As it is shown in [3], this equation permits a regularization of the vector field of directions of the gradient of u, i.e., there is a vector field z, $|z| \leq 1$, such that $u_t = div(z)$ and $\int_B z \cdot \nabla u = \int_B |\nabla u|$. Moreover, for each $t > 0$, $div(z(t)) \in L^p(B)$ if $u_0 \in L^p(B)$ for all $p \geq 1$. In this way, we initialize the steepest descent algorithm described in Section 3 with a regularized vector field θ. This again raises a question, namely, if this ad-hoc regularization is really needed or a regularization takes place with the algorithm itself, if we use an implicit numerical scheme to solve (10).

4. The bound $|u| \leq \|u_0\|_{L^\infty(B)}$ can be replaced by a constant depending on $\|u_0\|_{L^\infty(B)}$. The constraint that $u = u_0$ in B could be relaxed by adding a penalty term like $\int_B (u - u_0)^2$.

Similarly, we could add a penalty term to constraint θ to be near θ_0 inside B. In this case, we should regularize θ_0 in B using the equation described in Remark 3. This type of approach is addressed in the work of Chan and Shen mentioned before [9].

5. In practice, functional (10) is used to interpolate shapes, i.e., to interpolate level sets. The image is decomposed into upper level sets $[u_0 \geq \lambda]$, which are interpolated using (10) to produce the level sets $X_\lambda u$ of the function u which reconstructed inside Ω by using the reconstruction formula

$$u(x) = \sup\{\lambda : x \in X_\lambda u\}.$$

In principle, (10) could be used directly to interpolate functions. But, discontinuities of the image have a contribution to the energy which is proportional to the jump. This gives different weights to discontinuities of different sizes and, as a consequence, they are not treated in the same manner. This is not reasonable if we want to interpolate the shapes of the image, independently of their contrast. When taking level sets, we treat all shapes equally, and the parameters of the functional should only weight geometric quantities (like length, total curvature) and decide which interpolation is taken as a function of them. Which are the precise geometric quantities is not known precisely. The functional was introduced on a heuristic basis, but relaxations may occur as they occur in (16), where $div(\frac{\nabla u}{|\nabla u|})$ may represent $\frac{perimeter}{area}$ when computed on a flat region ([3, 36]). This requires further study and we shall pursue it elsewhere.

6. The choice made in Remark 5 of decomposing the image u_0 into upper level sets, interpolating them and reconstructing the function u, introduces a lack of symmetry. Indeed, we are giving more weight to upper level sets than to lower level sets. This can be seen in Figure 3. Figure 3.a displays the image to be interpolated. It is clear that several reasonable solutions are possible and no one of them is preferable to the others. The choice we made gives Figure 3.b as solution, favoring that the object whose level is 210 goes above the object whose level is 0. But, in that case, the "true" information is lacking and we selected one of the possible reasonable solution.

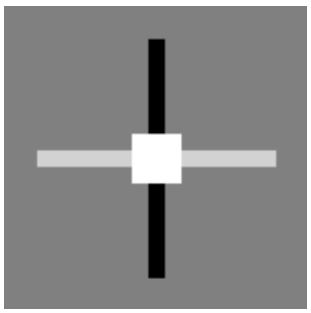


Figure 3.a.

Figure 3.b.

2.3 Interpolating gray values along the integral curves of a vector field

Our purpose in this section is to further discuss the term

$$\int_{\tilde{\Omega}} |\nabla u| - \theta \cdot \nabla u. \tag{17}$$

in functional (10) (see also [24] for a related, L_2 and Poisson-equation based, approach of gray value reconstruction from image gradients). We shall see that (when θ is known), when minimizing (17), we are constructing the function u whose values on the boundary are given and whose direction of the gradient is given by θ. We shall discuss this from a general point of view. Thus, suppose that Ω is an open bounded domain with a Lipschitz boundary and $\varphi \in L^\infty(\partial\Omega)$. Let $\nu : \Omega \to I\!R^2$ be a vector field whose smoothness will be detailed below. We ask the following question: can we interpolate the boundary data φ along the integral curves of ν ? In the case discussed in last Section, $\nu = \theta^\perp$, and we propagate the boundary data φ along the integral curves of $\theta^\perp$. Heuristically, θ is orthogonal to the level lines of u. Coming back to our general discussion, we want to construct a function $u : \Omega \to I\!R$ such that $u|_{\partial\Omega} = \varphi$ and u being constant along the integral curves of ν, i.e., the solutions of the system of ordinary differential equations

$$\frac{dX}{dt} = \nu(X). \tag{18}$$

This amounts to say that

$$\nu \cdot \nabla u = 0, \tag{19}$$

a first order transport equation whose characteristic curves are the solution of the system (18). Let us discuss the difficulties posed by this formulation. First of all, existence and uniqueness of solutions of (18) is guaranteed when ν is a Lipschitz vector field, a very strong regularity assumption, which excludes any singularity for ν. More general existence results have been obtained in [13] via the study of transport equations, indeed, via formulations analog to (19). Typically, they are assuming that ν is in some Sobolev space like $\in W^{1,1}_{loc}(I\!R^N)$, with some other integrability assumptions, and $div(\nu) \in L^\infty(I\!R^N)$. These results have been further extended in [12], [30]. In particular, P.L. Lions in [30] proves a.e. existence of solutions of (18) for vector fields which are piecewise in $W^{1,1}$ in a precise sense defined by the author. As observed in these papers, it is not known if the previous result is true for BV vector fields. On the other hand, even for a vector field in $W^{1,1}$, for which we have existence a.e. of solutions of (18), the problem of constructing u satisfying (19) and such that $u|_{\partial\Omega} = \varphi$ is not obvious. Indeed, consider a smooth vector field ν defined on a simple domain, like $D := \{x \in I\!R^2 : |x| \leq 1\}$ and suppose that the integral curves of ν are curves that foliate D and such that at any point of ∂D we start a curve that ends in another point of ∂D. Then the only possibility to extend u to Ω so that $u|_{\partial\Omega} = \varphi$ in a classical sense is that φ takes the same values at the beginning and endpoints of the integral curves of ν. A possibility to overcome this difficulty, would be to use the vanishing viscosity method, i.e., to solve the elliptic equation

$$\nu \cdot \nabla u_\epsilon + \epsilon \Delta u_\epsilon = 0, \tag{20}$$

and let $\epsilon \to 0$. Then, we hope the sequence u_ϵ to converge to some bounded function u which solves the problem in a distributional sense. We do not have further information on the regularity of u. On the other hand we do not know in which sense the boundary conditions hold.

Let us consider the problem from the algorithmic point of view, i.e., we want to design an effective algorithm to solve it. Since the problem may be ill-posed, because of incompatibility of boundary data joining two integral curves of the vector field ν, we propose a variational formulation of the problem. Let $\theta = \nu^\perp$. Assume that $|\theta| \leq 1$. If a solution exists, then θ should point in the normal direction to the level lines of u. We implicitly assume that θ should be constructed as the vector field normal to the level curves of u. Then, formally, $\theta \cdot \nabla u = |\nabla u|$. Thus, it seems reasonable to minimize the functional

$$F(u) = \int_\Omega |\nabla u| - \int_\Omega \theta \cdot \nabla u,$$

(exactly the one introduced above) defined in the set of functions of bounded variation $BV(\Omega)$ whose trace at the boundary is given by φ. Let us formally integrate by parts in the second term of $F(u)$ to obtain

$$F(u) = \int_\Omega |\nabla u| + \int_\Omega div(\theta) \cdot u - \int_{\partial\Omega} \theta \cdot \vec{n} u,$$

Since u, θ are known at the boundary, minimizing F amounts to minimize

$$E(u) = \int_\Omega |\nabla u| + \int_\Omega div(\theta) \cdot u.$$

Let us make precise the class of admissible functions where E is minimized. We assume that $div(\theta) \in L^1(\Omega)$ and $\varphi \in L^\infty(\partial\Omega)$. It seems reasonable to impose that the solution u is a bounded function with an L^∞ bound given by $\|\varphi\|_\infty$ (or a constant related to $\|\varphi\|_\infty$ and the size of Ω). Then the second integral in the definition of $E(u)$ is well defined. The first integral requires the use of the space of bounded variation functions. Thus our admissible class is $\mathcal{A} = \{u \in BV(\Omega) : |u(x)| \leq \|\varphi\|_\infty \text{ a.e. } u|_{\partial\Omega} = \varphi\}$. We propose

$$\begin{array}{l} \text{Minimize} \int_\Omega |\nabla u| + \int_\Omega div(\theta) \cdot u \\ u \in \mathcal{A} \end{array} \tag{21}$$

As it is well known ([21], [15]) the solution of this problem has to be understood in a weak sense as the solution of the problem

$$\begin{array}{l} \text{Minimize} \int_\Omega |\nabla u| + \int_\Omega div(\theta) \cdot u + \int_{\partial\Omega} |u - \varphi| dH^1 \\ u \in BV(\Omega) \\ |u| \leq \|\varphi\|_\infty. \end{array} \tag{22}$$

Then we have the following result.

Theorem 1 *Let $\theta \in L^1_{loc}(\Omega)^2$, with $div(\theta) \in L^1(\Omega)$, $\varphi \in L^\infty(\partial\Omega)$. Then there is a function $u \in BV(\Omega)$ such that $|u(x)| \leq \|\varphi\|_\infty$ a.e. minimizing (22).*

Proof. The result is contained in ([21]), Theorem 1.4.

This clarifies the role of the term (17) in (10).

3 Numerical experiments

To minimize (21) we use the steepest descent method. For that, we formally compute the Euler-Lagrange equation for u, namely,

$$-div\left(\frac{\nabla u}{|\nabla u|}\right) + div(\theta) = 0 \tag{23}$$

supplemented with Dirichlet boundary conditions for u. In practice, we use the evolution equation

$$u_t = div\left(\frac{\nabla u}{|\nabla u|}\right) - div(\theta)$$

with Dirichlet boundary data and initial condition constructed as an ad-hoc interpolation that will be corrected by the equation. General existence results which can be adapted to this case can be found in ([4]). Note that the vector field θ is assumed to be known in this case. This limits the usefulness of this model. But we present some experiments below to illustrate the role of this term.

To minimize the functional (10) we used the steepest descent method. For that, we formally compute the Euler-Lagrange equations for (θ, u). The equations for θ are

$$\nabla_\theta E(\theta, u) = -p\nabla[(a + b|\nabla k * u|)|div(\theta)|^{p-2}div(\theta)] - \nabla u\chi_\Omega - \nabla u_0\chi_B = 0 \quad \text{in } \tilde{\Omega}. \tag{24}$$

The equation for u is

$$\nabla_u E(\theta, u) = -div\left(k * g\frac{\nabla k * u}{|\nabla k * u|}\right) - \alpha div\left(\frac{\nabla u}{|\nabla u|}\right) + div(\theta) = 0 \quad \text{in } \Omega, \tag{25}$$

where $g = b|div(\theta)|^p$. In our experiments, we take k a Gaussian kernel with small variance, say one or two pixels. In practice, one can also dismiss the kernel k. These equations have to be complemented with the corresponding boundary conditions for θ and u specified by the admissible class, i.e., we specify the normal component of θ in $\partial\tilde{\Omega}$ and the Dirichlet boundary condition for u in $\partial\Omega$, since $u = u_0$ in B. Thus, we solve the evolution problems

$$\theta_t = -\nabla_\theta E(\theta, u)$$

and

$$u_t = -\nabla_u E(\theta, u),$$

supplemented with the corresponding boundary data and initial conditions. The initial conditions are ad-hoc interpolations, for instance, we can take u inside Ω as the average value of u_0 in B, θ inside Ω being the direction of the gradient of u. One can also take a geodesic propagation inside Ω of the values of u_0 in B, with θ being again the direction of the gradient of u. In the experiments below, this algorithm is used to interpolate level sets. In this, we follow the approach in [31], [32]. The image in B is decomposed into level sets and we get a family of binary images $u_{0\lambda} = \chi_{[u_0 \geq \lambda]}$, $\lambda = 0, 1, 2, ..., 255$. These functions are interpolated inside Ω and we obtain a family of level sets X_λ. Then the function u is reconstructed using the reconstruction formula

$$u(x) = \sup\{\lambda \in \{0, 1, ..., 255\} : x \in X_\lambda\}.$$

With this approach, we diminish the diffusive effects of the above algorithm and we better capture the shapes and discontinuities on the interpolated image. The constraints on θ and $\|u\|_\infty$ can be introduced after each iteration of the above equations.

Figure 4.a.

Figure 4.b.

Figure 4.c.

Let us describe the experiments. First, in Figure 4 we display some experiment to illustrate functional (21). Figure 4.a displays the full image without the hole. Figure 4.b displays the image with the hole. The vector field θ has been computed on Figure 4.a and we see in Figure 4.c the result of interpolating the gray level knowing the vector field inside Ω. We see that the shape of the eye is recovered but not the gray level. This is not a surprise since the gray level

inside the eye cannot be recovered from the gray level on the boundary of Ω. The algorithm is able to capture the shapes inside the eye by integrating the vector field θ.

In the following experiments we show the results of the joint interpolation of gray level and the vector field of directions using functional (10). The experiments have been done with $p = 1$ and/or $p = 2$. The results are quite similar. Unless explicitly stated, we display the results obtained with $p = 1$. Figure 5.a displays an image made of four circles covered by a square. In Figure 5.b we display the result of the interpolation. In Figure 6.a, we display an example where the hole is not simply connected. The interpolation is displayed in Figure 6.b. Figure 7.a is the image of Lena with two holes, a lower one in the hat and an upper one. Figure 7.b displays the result of the interpolation. Figure 8 displays a zoom of the region around the lower hole. In Figure 9.a we display a level set of u_0 corresponding to the region around the lower hole. Figures 9.b and 9.c display the corresponding interpolation with $p = 1$ and $p = 2$, respectively. Figure 10.a displays an image with text to be removed. Figure 10.b displays the corresponding reconstruction result. Figure 11.a displays a portion of an image with text. Figure 11.b displays the corresponding reconstruction result, obtained with $p = 2$.

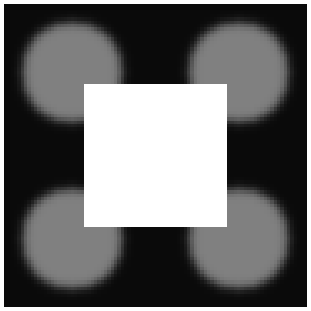


Figure 5.a.

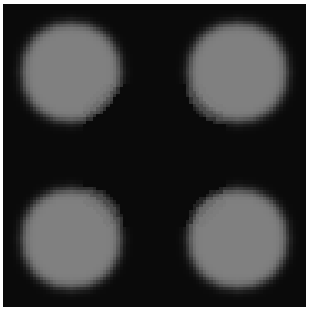


Figure 5.b.

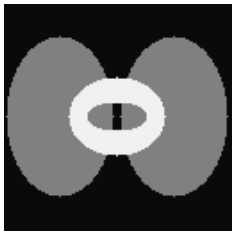


Figure 6.a.

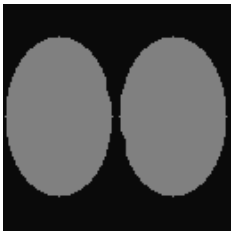


Figure 6.b.

Figure 7.a.

Figure 7.b.

Figure 8.a.

Figure 8.b.

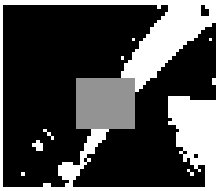


Figure 9.a.

Figure 9.b.

Figure 9.c.

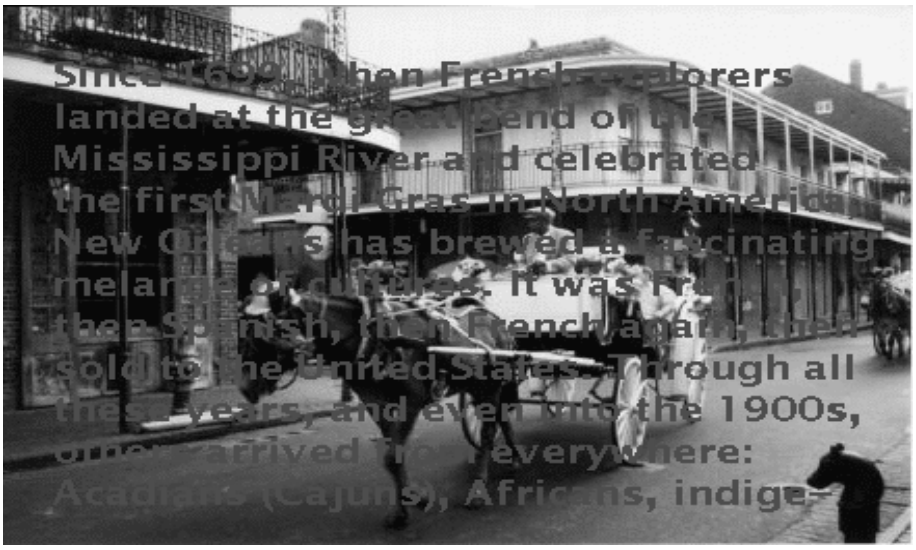


Figure 10.a.

Figure 10.b.

Figure 11.a.

Figure 11.b.

4 Concluding remarks

In this paper we have proposed a formal variational approach for filling-in regions of missing data in still images. The basic idea is to smoothly extend inside the hole both the vector field obtained from the image gradient and the corresponding gray values. We have presented a number of examples and showed theoretical results regarding the proposed formulation.

A number of research directions are suggested by the work here presented. First of all, we need to complement this algorithm by a technique capable of filling-in textured regions. Secondly, the extension of the framework to the filling-in of other type of missing imagery data is of great interest for a number of applications. Last, we would like to study these ideas for interpolation in video data. These topics will be the subject of subsequent reports.

Appendix: Existence of minimizers

Recall that $\tilde{\Omega} = \overline{\Omega} \cup B$ is an open bounded set whose boundary is Lipschitz. For simplicity, let us define the class $\mathcal{B}$ of admissible pairs (θ, u) where $\theta \in W^{1,p}(div, \tilde{\Omega})$, $u \in BV(\tilde{\Omega}, B, u_0)$, $|\theta| \leq 1$, $|u| \leq \|u_0\|_{L^\infty(B)}$ and $\theta \cdot n|_{\partial\tilde{\Omega}} = \theta_0 \cdot n|_{\partial\tilde{\Omega}}$.

Theorem 2 *If $p > 1$, there is a minimum $(\theta, u) \in \mathcal{B}$ for the problem (10).*

Proof. Let us denote by $E(\theta, u)$ the energy defined in (10). Let (θ_n, u_n) be a minimizing sequence for $E(\theta, u)$. Since

$$\int_{\tilde{\Omega}} |\nabla u_n| - \theta_n \cdot \nabla u_n \geq 0,$$

and $E(\theta_n, u_n)$ is bounded, we obtain that

$$\int_{\tilde{\Omega}} |div(\theta_n)|^p$$

is bounded. Since $|\theta_n| \leq 1$, we have that θ_n is weakly relatively compact in all spaces $L^q(\tilde{\Omega})^2$ for all $1 \leq q < \infty$ and we may assume that $\theta_n \to \theta$ weakly in $L^q(\tilde{\Omega})^2$ for all $1 \leq q < \infty$ and in $W^{1,p}(div, \tilde{\Omega})$. Now, integrating by parts the term $\int_{\tilde{\Omega}} \theta_n \cdot \nabla u_n$, we obtain

$$\begin{aligned} \int_{\tilde{\Omega}} \theta_n \cdot \nabla u_n &= -\int_{\tilde{\Omega}} div(\theta_n) u_n + \int_{\partial\tilde{\Omega}} [\theta_n, n] u_n \\ &= -\int_{\tilde{\Omega}} div(\theta_n) u_n + \int_{\partial\tilde{\Omega}} [\theta_0, n] u_0 \end{aligned}$$

The integration by parts is possible by results of Anzellotti ([2]) given above. From the above identity, we obtain

$$|\int_{\tilde{\Omega}} \theta_n \cdot \nabla u_n| \leq \|div(\theta_n)\|_p \|u_n\|_{p'} + \int_{\partial\tilde{\Omega}} |u_0|$$

since $u_n = u_0$ in $\partial\tilde{\Omega}$, where p' is the exponent conjugated to p. Since we minimize the energy $E(\theta, u)$ for functions with an L^∞ bound, we obtain that

$$|\int_{\tilde{\Omega}} \theta_n \cdot \nabla u_n|$$

is uniformly bounded in n. The consequence of this observation is that

$$\int_{\tilde{\Omega}} |\nabla u_n|$$

is also bounded. Then, modulo a subsequence,we may assume that u_n converges to some function u in $L^1(\tilde{\Omega})$. Note that $u \in BV(\tilde{\Omega}, B, u_0)$. Since we have an L^∞ bound on u_n, we also have that u_n converges to u in $L^q(\tilde{\Omega})$ for all $1 \leq q < \infty$. Then $\nabla k * u_n \to \nabla k * u$ uniformly in $\tilde{\Omega}$. In particular, we obtain

$$\int_{\tilde{\Omega}} |div(\theta)|^p (a + b|\nabla k * u|) dx \leq \liminf_n \int_{\tilde{\Omega}} |div(\theta_n)|^p (a + b|\nabla k * u_n|) dx$$

and

$$\int_{\tilde{\Omega}} |\nabla u| \leq \liminf_n \int_{\tilde{\Omega}} |\nabla u_n|.$$

Finally, since $div(\theta_n)$ weakly converges to $div(\theta)$ in $L^p(\tilde{\Omega})$ and $u_n \to u$ in $L^{p'}(\tilde{\Omega})$, passing to the limit in

$$\int_{\tilde{\Omega}} \theta_n \cdot \nabla u_n = -\int_{\tilde{\Omega}} div(\theta_n) u_n + \int_{\partial\tilde{\Omega}} [\theta_0, n] u_0$$

we get that $\int_{\tilde{\Omega}} \theta_n \cdot \nabla u_n$ converges to

$$-\int_{\tilde{\Omega}} div(\theta) u + \int_{\partial\tilde{\Omega}} [\theta_0, n] u_0 = \int_{\tilde{\Omega}} \theta \cdot \nabla u.$$

Thus, collecting all these facts, we obtain that

$$E(\theta, u) \leq \liminf_n E(\theta_n, u_n).$$

The pair (θ, u) is a minimum of E in the class of admissible functions for this functional.

Acknowledgments

We thank S. Betelu, A. Bertozzi, T. Chan, C. Kenney, P.L. Lions, J.M. Morel, S. Osher, E. Simoncelli, and J. Shen for interesting conversations on image inpainting and filling-in. This work was partially supported by a grant from the Office of Naval Research ONR-N00014-97-1-0509, the Office of Naval Research Young Investigator Award, the Presidential Early Career Awards for Scientists and Engineers (PECASE), a National Science Foundation CAREER Award, by the National Science Foundation Learning and Intelligent Systems Program (LIS), and the TMR European project "Viscosity Solutions and their applications," reference FMRX-CT98-0234.

References

[1] L. Ambrosio, N. Fusco and D. Pallara, *Functions of Bounded Variation and Free Discontinuity Problems*, Forthcoming book.

[2] G. Anzellotti, "Pairings between measures and bounded functions and compensated Compactness," *Ann. di Mat. Pura ed Appl.* **135**, 293-318, 1993.

[3] F. Andreu, C. Ballester, V. Caselles, J. M. Mazón, "Minimizing Total Variation flow," *Differential and Integral Equations*, to appear.

[4] F. Andreu, C Ballester, V. Caselles and J.M. Mazón, "The Dirichlet problem for the Total Variation flow," preprint.

[5] G. Bellettini, G. Dal Maso and M. Paolini, "Semicontinuity and relaxation properties of a curvature depending functional in 2D," *Ann. Scuola Normale Sup. di Pisa, Cl. Sci.* **20(4)**, pp. 247-297, 1993.

[6] M. Bertalmio, G. Sapiro, C. Ballester and V. Caselles, "Image inpainting," *University of Minnesota IMA TR*, December 1999 (available at www.ima.umn).

[7] M. Bertalmio, G. Sapiro, C. Ballester and V. Caselles, "Image inpainting," to appear in *Computer Graphics, SIGGRAPH 2000*, July 2000.

[8] C. Braverman. *Photoshop retouching handbook*. IDG Books Worldwide, 1998.

[9] T. Chan and J. Shen, "Mathematical models for local deterministic inpaintings," *UCLA CAM Report* **00-11**, March 2000 (available at www.math.ucla.edu).

[10] R. Dautray and J.L. Lions, *Analyse Mathématique et Calcul Numérique pour les Sciences et les Techniques, Tome 2*, Masson, 1985.

[11] F. Demengel, *Introduction aux Équations aux Derivées Partielles Elliptiques*, Diderot Ed. 1999.

[12] B. Desjardins, "A few remarks on ordinary differential equations," *Comm. in Partial Diff. Equations* **21**, pp. 1667-1703, 1996.

[13] R. J. DiPerna and P. L. Lions, "Ordinary differential equations, transport theory and Sobolev spaces," *Inventiones Math.* **98**, pp. 511-547, 1989.

[14] A. Efros and T. Leung, "Texture synthesis by non-parametric sampling," *Proc. IEEE International Conference Computer Vision*, pp. 1033-1038, Corfu, Greece, September 1999.

[15] I. Ekeland and R. Temam, *Convex Analysis and Variational Problems*, North Holland, Amsterdam, 1976.

[16] G. Emile-Male. *The Restorer's Handbook of Easel Painting*. Van Nostrand Reinhold, New York, 1976.

[17] L.C. Evans and R.F. Gariepy, *Measure Theory and Fine Properties of Functions*, Studies in Advanced Math., CRC Press, 1992.

[18] G. P. Galdi, *An Introduction to the Mathematical Theory of the Navier-Stokes Equations, Volume I*, Springer Verlag, 1994.

[19] D. Heeger and J. Bergen. "Pyramid based texture analysis/synthesis," *Computer Graphics*, pp. 229-238, SIGGRAPH 95, 1995.

[20] P. G. de Gennes and J. Prost, *The Physics of Liquid Crystals*, Oxford Science Publications, 1998.

[21] M. Giaquinta, G. Modica and J. Soucek, "Functionals with linear growth in the calculus of variations I," *Comment. Math. Univ. Carolina* **20**, pp. 143-156, 1979.

[22] A. Hirani and T. Totsuka. "Combining frequency and spatial domain information for fast interactive image noise removal," *Computer Graphics*, pp. 269-276, SIGGRAPH 96, 1996.

[23] L. Joyeux, O. Buisson, B. Besserer, S. Boukir. "Detection and removal of line scratches in motion picture films," *Proceedings of CVPR'99, IEEE Int. Conf. on Computer Vision and Pattern Recognition,* Fort Collins, Colorado, USA, June 1999.

[24] C. Kenney and J. Langan, "A new image processing primitive: Reconstructing images from modified flow fields," preprint, University of California, Santa Barbara.

[25] D. King. *The Commissar Vanishes.* Henry Holt and Company, 1997.

[26] R. Kohn and R. Temam, "Dual spaces of stresses and strains, with applications to Hencky plasticity," *Appl. Math. Optimization* **10** (1983), 1-35.

[27] A.C. Kokaram, R.D. Morris, W.J. Fitzgerald, P.J.W. Rayner. "Detection of missing data in image sequences," *IEEE Transactions on Image Processing* **11(4)**, pp. 1496-1508, 1995.

[28] A.C. Kokaram, R.D. Morris, W.J. Fitzgerald, P.J.W. Rayner. "Interpolation of missing data in image sequences," *IEEE Transactions on Image Processing* **11(4)**, pp. 1509-1519, 1995.

[29] A. Lichnewski and R. Temam, "Pseudosolutions of the time dependent minimal surface problem," *Journal of Differential Equations* **30**, pp. 340-364, 1978.

[30] P.L. Lions, "Sur les équations différentielles ordinaires et les équations de transport," *Comptes Rendus Acad. Sciences* **326**, pp. 833-838, 1998.

[31] S. Masnou, *Filtrage et Desocclusion d'Images par Méthodes d'Ensembles de Niveau*, Thèse, Université Paris-Dauphine, 1998.

[32] S. Masnou and J.M. Morel, "Level lines based disocclusion," *5th IEEE International Conference on Image Processing*, Chicago, Illinois, October 4-7, 1998.

[33] M. Nitzberg, D. Mumford, and T. Shiota, *Filtering, Segmentation, and Depth*, Springer-Verlag, Berlin, 1993.

[34] L. Rudin, S. Osher and E. Fatemi. "Nonlinear total variation based noise removal algorithms," *Physica D* **60**, pp. 259-268, 1992.

[35] E. Simoncelli and J. Portilla. "Texture characterization via joint statistics of wavelet coefficient magnitudes," *5th IEEE Int'l Conf. on Image Processing*, Chicago, IL. Oct 4-7, 1998.

[36] D. Strong and T. Chan, "Exact solutions to Total Variation regularization problems," *UCLA CAM Report* **96-41**, October 1996.

[37] R. Temam, "On the continuity of the trace of vector functions with bounded deformation," *Appl. Analysis* **11**, pp. 291-302, 1993.

[38] S. Walden. *The Ravished Image.* St. Martin's Press, New York, 1985.

[39] W. P. Ziemer, *Weakly Differentiable Functions*, GTM 120, Springer Verlag, 1989.

Fast Computation of Weighted Distance Functions and Geodesics on Implicit Hyper-Surfaces

Facundo Mémoli
Instituto de Ingeniería Eléctrica
Universidad de la República
Montevideo, Uruguay
memoli@iie.edu.uy

Guillermo Sapiro*
Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN 55455
guille@ece.umn.edu

March 2001

Abstract

An algorithm for the computationally optimal construction of intrinsic weighted distance functions on implicit hyper-surfaces is introduced in this paper. The basic idea is to approximate the intrinsic weighted distance by the Euclidean weighted distance computed in a band surrounding the implicit hyper-surface in the embedding space, thereby performing all the computations in a Cartesian grid with classical and computationally optimal numerics. Based on work on geodesics on Riemannian manifolds with boundaries, we bound the error between the two distance functions. We show that this error is of the same order as the theoretical numerical error in computationally optimal, Hamilton-Jacobi based, algorithms for computing distance functions in Cartesian grids. Therefore, we can use these algorithms, modified to deal with spaces with boundaries, and obtain also for the case of intrinsic distance functions on implicit hyper-surfaces a computationally optimal technique. The approach can be extended to solve a more general class of Hamilton-Jacobi equations defined on the implicit surface, following the same idea of approximating their solutions by the solutions in the embedding Euclidean space. The framework here introduced thereby allows to perform the computations on a Cartesian grid with computationally optimal algorithms, in spite of the fact that the distance and Hamilton-Jacobi equations are intrinsic to the implicit hyper-surface. For other surface representations like triangulated or unorganized points ones, the algorithm here introduced can be used after simple pre-processing of the data.

*Corresponding author

Contents

1 Introduction

Computing distance functions has a number of applications in numerous areas including mathematical physics, image processing, computer vision, robotics, computer graphics, computational geometry, optimal control, and brain research. In addition, having the distance function from a seed to a target point, it is straightforward to compute the corresponding geodesic path, since this is given by the gradient direction of the distance function, back propagating from the target toward the seed (see for example [17]). Geodesics are used for example for path planning in robotics [34], brain flattening and brain warping in computational neuroscience [55, 56, 58, 59, 66], crests, valleys, and silhouettes computations in computer graphics and brain research [7, 29, 60], mesh generation [62], and many applications in mathematical physics. Distance functions are also very important in optimal control [57] and computational geometry for computations such a Voronoi diagrams and skeletons [47]. It is then of extreme importance to develop efficient techniques for the accurate and fast computations of distance functions. It is the goal of this paper to present an accurate and computationally optimal technique for the computation of intrinsic weighted distance functions on implicit hyper-surfaces.[1] It is well-know already, and it will be further detailed below, that this weighted distances can be obtained as the solution of simple Hamilton-Jacobi equations. We will also show that the framework here presented can be applied to a larger class of Hamilton-Jacobi equations defined on implicit surfaces. We also discuss the application of our proposed framework to other, non-implicit, surface representations.

1.1 Distance Function Computation and its Hamilton-Jacobi Formulation

Before proceeding, let us first formally define the concept of *intrinsic* weighted distances on *implicit* hyper-surfaces. Let $\mathcal{S}$ be a (codimension 1) hyper-surface in $\mathbb{R}^d$ defined as the zero level set of a function $\psi : \mathbb{R}^d \to \mathbb{R}$. That is, $\mathcal{S}$ is given by $\{x \in \mathbb{R}^d : \psi(x) = 0\}$. We assume from now on that ψ is a signed distance function to the surface $\mathcal{S}$. (This is a not a limitation, since as we will discuss later, both explicit and implicit representations can be transformed into this form.) Our goal is, for a given point $p \in \mathcal{S}$, to compute the intrinsic g-weighted distance function $d^g_{\mathcal{S}}(p, x)$ for all desired points $x \in \mathcal{S}$.[2] Note that we are referring to the intrinsic g-distance, that is, the geodesic distance on the Riemannian manifold $(\mathcal{S}, g^2 I\!I)$ ($I\!I$ stands for the $d \times d$ identity matrix) and not on the embedding Euclidean space. For a given weight g defined on the surface (we are considering only isotropic metrics for now), the g-distance on $\mathcal{S}$ (that coincides with the geodesic distance of the Riemannian manifold $(\mathcal{S}, g^2 I\!I)$) is given by

$$d^g_{\mathcal{S}}(p, x) \stackrel{\triangle}{=} \inf_{\mathcal{C}_{px}[\mathcal{S}]} \{\mathbf{L}_g(\mathcal{C})\} \tag{1}$$

where

$$\mathbf{L}_g\{\mathcal{C}\} \stackrel{\triangle}{=} \int_a^b g\left(\mathcal{C}(l)\right) \, \|\dot{\mathcal{C}}(l)\| \, dl \tag{2}$$

[1] Although all the examples in this paper are going to be reported for 3D surfaces, the theory is valid for general dimension hyper-surfaces, and it will be presented in this generality. A number of applications deal with higher dimensions. For example, for the general theory of harmonic maps, in order to deal with maps onto general open surfaces, it is necessary to have this notion of intrinsic distance [40]. In addition, higher dimensions might appear in motion planning, when explicitly assuming that the robot is not modeled by a point, thereby adding additional constraints to its movements.

[2] This can certainly be extended to any subset of $\mathcal{S}$.

is the weighted *length functional* defined for piecewise C^1 curves $\mathcal{C} : [a, b] \to \mathcal{S}$, and $C_{px}[\mathcal{S}]$ denotes the set of curves piecewise C^1 joining p to x, traveling on $\mathcal{S}$. In general we will consider the definition to be valid for any $\tilde{g}$ defined over the domain that the curve may travel through .

We need to compute this distance when all the concerning objects are represented in discrete form in the computer. Computing minimal weighted distances and paths in graph representations is an old problem that has been optimally solved by Dijkstra [20]. Dijkstra showed an algorithm for computing the path in $O(n \log n)$ operations, where n is the number of nodes in the graph. The weights are given on the edges connecting between the graph nodes, and the algorithm is computationally optimal. In theory, we could use this algorithm to compute the weighted distance and corresponding path on polygonal (not implicit) surfaces, being the vertices the graph nodes and the edges the connections between them (see [32]). The problem is that this algorithm is limited to travel on the graph edges, giving only a first approximation of the true distance. Moreover, Dijkstra's is not a consistent algorithm: it will not converge to the true desired distance when the graph and grid is refined [41, 42]. The solution to this problem, limited to Cartesian grids, was developed in [26, 50, 51, 57] (and recently extended by Osher and Helmsen, see [44]). Tsitsiklis first described an optimal-control type of approach, while independently Sethian and Helmsen both developed techniques based on upwind numerical schemes. The solution presented by these authors is consistent and converges to the true distance [48, 57], while keeping the same optimal complexity of $O(n \log n)$. This work was later extended in [31] for triangulated surfaces (see also [7, 35] for related works on numerics on non-Cartesian grids). We should note that the algorithm developed in [31] is currently developed only for triangulated surfaces with acute triangles. Therefore, before the algorithm can be applied, as an initialization step the surfaces have to be pre-processed to remove all obtuse triangles or other polygons present in the representation [30]. Following [51], we call to these algorithms *fast marching*.

The basic idea behind the computationally optimal techniques for finding accurate weighted distances, fast marching algorithms, is to note that the distance function holds a Hamilton-Jacobi Partial Differential Equation (PDE) in the viscosity sense; see for example [37, 49] for the general topic of distance functions on Riemannian manifolds (and a nice mathematical treatment), and [12, 22, 30, 43, 45, 51] for the planar (and more intuitive) case. This Hamilton-Jacobi is given by

$$\|\nabla_{\mathcal{S}} d^g_{\mathcal{S}}\| = g \tag{3}$$

where $\nabla_{\mathcal{S}}$ is the gradient intrinsic to the surface, and $d^g_{\mathcal{S}}$ is the g-distance from a given seed point to the rest of the manifold.[3]

That is, we can transform the problem of optimal distance computation into the problem of solving a Hamilton-Jacobi equation (recall that g is known, it is the given weight), also known as the Eikonal equation. In order to solve this equation, the current state of knowledge permits to accurately and optimally (in a computational sense) find (weighted) distances on Cartesian grids as well as on particular triangulated surfaces (after some pre-processing, namely the elimination of obtuse triangles, see [6, 31]). The goal of this paper is to extend this to implicit hyper-surfaces. In other words, we will show how to solve the above Eikonal equation for implicit hyper-surfaces $\mathcal{S}$.

Recall that although all the applications in this paper will be presented for 3D surfaces, the theory is valid for any d-dimensional hyper-surfaces, and will then be presented in this generality.

[3]Note that $\nabla_{\mathcal{S}}$ and $d^g_{\mathcal{S}}$ become the classical gradient and distance respectively for Euclidean spaces.

1.2 Distance Function and Geodesics on Implicit Surfaces

The motivations behind extending the distance map calculation to implicit surfaces are numerous: **a)** in many applications, surfaces are already given in implicit form, e.g., [10, 13, 15, 24, 44, 45, 52, 64, 60], and there is then a need to extend to this important representation the fast techniques previously mentioned. We could of course triangulate the implicit surface, eliminate obtuse triangles, and then use the important algorithm proposed in [31]. This is not a desirable process in general when the original data is in implicit form, since it affects the distance computation accuracy due to errors from the triangulation, and also adds the computational cost of the triangulation itself, triangulation that might not be needed by the specific application. If for example all what it is needed is to compute the distance between a series of points on the surface, the computational cost added by the triangulation is unnecessary. Note that finding a triangulated representation of the implicit surface is of course dimensionality dependent, and adds the errors of the triangulation process. Moreover, accurate triangulations that ensure correctness in the topology are computationally expensive, and once again there is no reason to perform a full triangulation when we might be interested just in the intrinsic distance between a few points on the implicit surface. **b)** it is a general agreement that the work on implicit representations and Cartesian grids is more robust when dealing with differential characteristics of the surface and partial differential equations on it. Numerical analysis on Cartesian grids is much more studied and supported by fundamental results than the work on polygonal surfaces. It is even recognized that there is no consensus about how to compute basic differential quantities over a triangulated surface, see for example [21], although there is quite an agreement for implicit surfaces. Moreover, representing an hyper-surface with structured elements such as triangles is certainly difficult for dimensions other than 2 or 3. **c)** if the computation of the distance function is just a part of a general algorithm for solving a given problem, it is not elegant, accurate, nor computationally efficient to go back and forth from different representations of the surface.

Before proceeding, we should note that although the whole framework and theory is here developed for implicit surfaces, it is valid for other surface representations as well after simple pre-processing. This will be discussed later in the paper (§5). Moreover, we will lately assume that the embedding is a distance function. This is not a limitation, since many algorithms exist to transform a generic embedding function into a distance one; see also §5. Therefore, the framework here presented can be applied both to implicit (naturally) and other surface representations like triangulated ones.

In order to compute intrinsic distances on surfaces, a small but important number of techniques have been reported in the literature. As mentioned before, in a very interesting work Kimmel and Sethian extended the fast marching algorithm to work on triangulated surfaces. In its current version, this approach can only be used when dealing with 3D triangulated surfaces and its extension to deal with higher dimensions seems very involved. Moreover, it can only correctly handle acute triangulations (thereby requiring a pre-processing step). And of course, it doesn't apply to implicit surfaces without some pre-processing (a triangulation).

Another very interesting approach to computing intrinsic distances, this time working with implicit surfaces, was introduced in [15]. This will be further described below, but before that let's make some comments on it. First, this is an evolutionary/iterative approach, whose steady state gives the solution to the corresponding Hamilton-Jacobi. Therefore, this approach it is not computationally optimal for the class of Hamilton-Jacobi equations discussed in this paper.[4] Second, very careful discretization must be done to the equation proposed in [15] do to the presence

[4]The general framework introduced in [15] is applicable beyond the Hamilton-Jacobi equations discussed in this

of intrinsic jump functions that might change the zero level-set (i.e., the surface). On the other hand, the numerical implementation is not necessary done via the utilization of *monotone schemes*, as required by our approach and all the fast marching techniques previously mentioned (thereby having a theoretical error $\Theta(\sqrt{\Delta x})$ [18]), and better accuracy might then be obtained.

In order to compute the intrinsic distance on an implicit surface, we must then solve the corresponding Hamilton-Jacobi equation presented before. In order to do this in a computationally efficient way, we need to extend the fast marching ideas in [26, 44, 50, 51, 57], which assume a Cartesian grid, to work in our case. Since an implicit surface is represented in a Cartesian grid, corresponding to the embedding function, the first and most intuitive idea is then to attempt to solve the *intrinsic Eikonal* using the *fast marching* technique. The first step towards our goal is to express all the quantities in the intrinsic Eikonal by its *implicit-extended* representations. What we mean is that the *intrinsic* problem (we consider $g = 1$ for simplicity of exposition)

$$\begin{cases} \|\nabla_{\mathcal{S}}\, d_{\mathcal{S}}(p)\| = 1 \ \text{ for } p \in \mathcal{S} \\ d_{\mathcal{S}}(p) = 0. \end{cases} \tag{4}$$

with $p \in \mathcal{S}$ the seed point, is to be extended to all $\mathbb{R}^d$ (or at least to a band surrounding $\mathcal{S}$), and the derivatives are to be taken tangentially to $\{\psi = 0\}$. Considering then the projection of the Euclidean gradient onto the tangent space of $\mathcal{S}$ to obtain the intrinsic one, and denoting by $\hat{d}$ the Euclidean extension to the intrinsic distance $d_{\mathcal{S}}$, we have to numerically solve, in the embedding Cartesian grid, the equation

$$\begin{cases} \|\nabla \hat{d}(x)\|^2 - |\nabla \hat{d}(x) \cdot \nabla \psi(x)|^2 = 1 \ \text{ for } x \in \mathbb{R}^d \\ \hat{d}(l(p)) = 0. \end{cases} \tag{5}$$

where $l(p)$ is the ray through p normal to the level sets of ψ.

This is exactly the approach introduced in [15], as discussed above, to build-up the evolutionary approach, given by the following PDE:

$$\phi_t + sgn(\phi_0)\left(\sqrt{\|\nabla\phi\|^2 - |\nabla\phi \cdot \nabla\psi|^2} - 1\right) = 0 \tag{6}$$

where $\phi_0(x) = \phi(x, 0)$ is the initial value of the evolving function, generally a step-like function that tells inside from outside of the zero level-set. One then finds $\hat{d}(\cdot) = \phi(\cdot, \infty)$.

Of course, in order to obtain a computationally optimal approach, we want to solve the stationary problem (5), and not its iterative counterpart (6). It turns out that the basic requirements for the construction of a fast marching method, even with the recent extensions in [44], do not hold for this equation. This can be explicitly shown, and has also been hinted by Kimmel and Sethian in their work on geodesics on surfaces given as graphs of functions.[5]

To recap, the fast marching approach cannot be directly applied to the computation of intrinsic distances on implicit surfaces defined on a Cartesian grid (equation (5)), and the state of the art in numerical analysis for this problem says that in order to compute intrinsic distances one has either to work with triangulated surfaces or has to use the iterative approach mentioned above. The problems with both techniques were reported before, and it is the goal of this paper to present a third approach that addresses all these problems.

paper (see also [8, 16]). Here we limit the comparison between the techniques to the equations were both approaches are applicable.

[5] We have also benefited from private conversations with Stan Osher and Ron Kimmel to confirm this claim.

1.3 Our Contribution

The basic idea here presented is conceptually very simple. We first consider a small h offset of $\mathcal{S}$. That is, since the embedding function ψ is a distance function, with $\mathcal{S}$ as its zero level set, we consider all points x in $\mathbb{R}^3$ for which $|\psi(x)| \leq h$. This gives a region in $\mathbb{R}^d$ with boundaries. We then modify the (Cartesian) fast marching algorithm mentioned above for computing the distance transform inside this h-band surrounding $\mathcal{S}$. Note that here, all the computations are as in the works in [26, 50, 51, 57], in a Cartesian grid. We then use this Euclidean distance function as an approximation of the intrinsic distance on $\mathcal{S}$. In §2 we show that the error between these two distances, under reasonable assumptions on the surface $\mathcal{S}$, is of the same order as the numerical error introduced by the fast marching algorithms in [26, 50, 51, 57].[6] Therefore, when adapting these algorithms to work on Euclidean spaces with boundary, adaptation described in §3, we obtain an algorithm for the computation of intrinsic distances on implicit surfaces with the same simplicity, computational complexity, and accuracy than the optimal fast marching techniques for computing Euclidean distances on Cartesian grids.[7] In §3 we also explicitly discuss the numerical error of our proposed technique. Examples of the algorithm here proposed are given in §4. Since Osher and Helmsen have recently shown that the fast marching algorithm can be used to solve additional Hamilton-Jacobi equations, we show that the framework here proposed can be applied to equations from that class as well, this is done in §5. This section also discusses the use of the framework here presented for non-implicit surfaces. Finally, some concluding remarks are given in §6.

2 Distance Functions: Intrinsic vs. Extrinsic

The goal of this section is to present the connection between the intrinsic distance function and the Euclidean one computed inside a band surrounding the (implicit) surface. We will completely characterize the difference between these two functions, mainly based on results on shortest paths on manifolds with boundary. The results here presented will justify the use of the Cartesian fast marching algorithms also for the computation of intrinsic weighted distances on implicit surfaces.

Recall that we are dealing with a closed hyper-surface $\mathcal{S}$ in $\mathbb{R}^d$ represented as the zero level-set of a distance function $\psi : \mathbb{R}^d \to \mathbb{R}$. That is, $\mathcal{S} = \{\psi = 0\}$. Our goal is to compute a g-weighted distance map on this surface from a seed point $q \in \mathcal{S}$. Let

$$\Omega_h \stackrel{\triangle}{=} \bigcup_{x \in \mathcal{S}} B(x,h) \;=\; \{x \in \mathbb{R}^d : |\psi(x)| \leq h\}$$

be the *h-offset* of $\mathcal{S}$ (here $B(x,h)$ is the ball centered at x with radius h). It is well known that for a smooth $\mathcal{S}$, $\partial\Omega_h$ is also smooth if h is sufficiently small, see Appendix A for references. Ω_h is then a *manifold with smooth boundary.*

Our computational approach is based on approximating the solution of the *intrinsic* problem ($d^g_{\mathcal{S}}(p)$ is the intrinsic g-weighted distance on $\mathcal{S}$).

[6] In contrast with works such as [1, 46], where an offset of this form is just used to improve the complexity of the level-sets method, in our case the offset is needed to obtain a small error between the computed distance transform and the real intrinsic distance function, see next Section.

[7] Although in this paper we deal with the fast marching techniques, other techniques for computing distance functions on Cartesian grids, e.g., the fast technique reported in [11] for uniform weights, could be used as well, since the basis of our approach is the approximation of the intrinsic distance by an extrinsic one.

$$\begin{cases} \|\nabla_{\mathcal{S}} d^g_{\mathcal{S}}(p)\| = g \ \text{ for } p \in \mathcal{S} \\ d^g_{\mathcal{S}}(q) = 0. \end{cases} \tag{7}$$

by that of the *Euclidean* (or *extrinsic*) one:

$$\begin{cases} \|\nabla d^{\tilde{g}}_{\Omega_h}(p)\| = \tilde{g} \ \text{ for } p \in \Omega_h \\ d^{\tilde{g}}_{\Omega_h}(q) = 0. \end{cases} \tag{8}$$

where $\tilde{g}$ is a smooth *extension* of g in a domain containing Ω_h, and $d^{\tilde{g}}_{\Omega_h}(p)$ is the Euclidean $\tilde{g}$-weighted distance in Ω_h. Our goal is to be able to control $\|d^g_{\mathcal{S}} - d^{\tilde{g}}_{\Omega_h}\|_{L_\infty(\mathcal{S})}$ with h. Note that we have replaced the intrinsic gradient $\nabla_{\mathcal{S}}$ by the Euclidean one and the intrinsic distance $d^g_{\mathcal{S}}(p)$ on the surface by the Euclidean distance $d^{\tilde{g}}_{\Omega_h}(p)$ in Ω_h. We have then transformed the problem of computing an intrinsic distance into the problem of computing a distance in an Euclidean manifold with boundary.

We will show that under suitable (and likely) geometric conditions on $\mathcal{S}$ we can indeed control $\|d^g_{\mathcal{S}} - d^{\tilde{g}}_{\Omega_h}\|_{L_\infty(\mathcal{S})}$ with h. In order to materialize this, we first need to briefly discuss the extension $\tilde{g}$ and to review some basic background material on Riemannian manifolds with boundary.

2.1 The Extension of the Weight g

We require that $\tilde{g}|_{\mathcal{S}} = g$, and that $\tilde{g}$ is *smooth* within Ω_h. There are situations when one has a readily available extension, other where the extension has to be "invented." We call the former *natural extension* and the latter *general extension.* Both cases, as argued below, will provide smooth functions $\tilde{g}$.

In many applications the weight $g : \mathcal{S} \to I\!R$ depends on the curvature structure of the hyper-surface. Denoting $\mathbf{B}_{\mathcal{S}}(\cdot) : \mathcal{S} \to I\!R^{d\times d}$ the second fundamental form of $\mathcal{S}$, and $\Lambda\left(\mathbf{B}_{\mathcal{S}}(x)\right)$ the set of its eigenvalues, this means that

$$g(x) = F(\Lambda\left(\mathbf{B}_{\mathcal{S}}(x)\right)$$

where F is a given function. In this case it is utterly *natural* to take advantage of the implicit representation by noting that $\mathbf{B}_{\mathcal{S}}(x) = \mathbf{H}_{\psi|_{T_x\mathcal{S}}}(x)$ for $x \in \mathcal{S}$, where $\mathbf{H}_\psi$ is the Hessian of ψ and $T_x\mathcal{S}$ is the tangent space to $\mathcal{S}$ at x (see [36]). The *natural* extension then becomes

$$\tilde{g}(x) = F(\Lambda(\mathbf{H}_{\psi|_{T_x\mathcal{S}}}(x))), \ \ x \in \Omega_h \tag{9}$$

This extension is valid for $\{x \in I\!R^d : |\psi(x)| < 1/\mathcal{M}_{\mathcal{S}}\}$, where $\mathcal{M}_{\mathcal{S}}$ absolutely bounds all principal curvatures of $\mathcal{S}$, see Appendix A.

When the weight g cannot be directly extended to be valid for a tubular neighborhood of the hyper-surface, one has to do that in a pedestrian way. One such extension comes from propagating the values of g along the normals of $\mathcal{S}$ in a constant fashion, i.e.:

$$\tilde{g}(x) = g(\Pi_{\mathcal{S}}(x)), \ \ x \in \Omega_h \tag{10}$$

where $\Pi_{\mathcal{S}}(\cdot) : I\!R^d \to \mathcal{S}$ stands for the normal projection onto $\mathcal{S}$. This extension is well defined and smooth as long as there is a unique *foot* in $\mathcal{S}$ for every x in the domain of the extension Ω. Taking h sufficiently small we can guarantee that $\Omega \supset \Omega_h$ if $\mathcal{S}$ is smooth. See Appendix A for some details.

In practice this extension can be accomplished solving the equation [14]

$$\phi_t + sgn(\psi)\ \nabla\psi \cdot \nabla\phi = 0$$

with initial conditions given by any $\phi(\cdot, 0)$ such that $\phi(\cdot, 0)|_{\mathcal{S}} = g$. Then $\tilde{g}(\cdot) \stackrel{\triangle}{=} \phi(\cdot, \infty)$.

2.2 Shortest Paths and Distance Functions in Manifolds with Boundary

Since we want to approximate the problem of intrinsic distance functions by a problem of distance functions in manifolds with boundary, and to prove that the latter converges to the former, we need to review basic concepts on this subject. We will mainly include results from [2, 3, 61]. We are interested in the existence and smoothness of the geodesic curves on manifolds with boundary, since our convergence arguments below depend on these properties. We will assume throughout this section that $(\mathcal{M}, m)$ is a *connected* and *complete* Riemannian manifold with boundary (this will later become the h-offset Ω_h with the metric $\tilde{g}^2 I\!I$).

Deffinition 1 *Let $p, q \in \mathcal{M}$, then if $d_{\mathcal{M}}(\cdot,\cdot) : \mathcal{M} \times \mathcal{M} \to I\!R$ is the* distance function *in $\mathcal{M}$ (with its metric m), a* shortest path *between p and q is a path joining them such that its Riemannian length equals $d_{\mathcal{M}}(p, q)$.*

Now, since $\mathcal{M}$ is *complete*, for every pair of points p and q there exist a *shortest path* joining them, see [2]. The following results deals with the regularity of this shortest path.

Theorem 1 *Let $(\mathcal{M}, m)$ be a C^3 manifold with C^1 boundary B. Then any* shortest path *of $\mathcal{M}$ is C^1.*

When $(\mathcal{M}, m)$ is a flat manifold (i.e. $\mathcal{M}$ is a codimension 0 subset of $I\!R^d$ and the metric m is isotropic and constant), it is easy to see that any *shortest path* must be a straight line whenever it is in the interior of $\mathcal{M}$, and a shortest path of the boundary B when it is there. That will be the situation for us from now on.

It might seem a bit awkward that one cannot achieve higher regularity class than C^1 for the shortest paths, even by increasing the regularity of $\mathcal{M} \cup B$, but a simple counterexample will convince the reader. Think of $\mathcal{M}$ as $I\!R^2$ with the open unit disc removed, see Figure 1, and its Euclidean metric. The acceleration in all the open segment (AP) is $\vec{0}$, and in all the open arc (PQ) is $-\vec{e}_r$, that is, it points inwards, and has modulus 1. That is, even in most simple examples, C^2 regularity is not achievable. It is, however, very easy to check that in this case $\dot{\gamma}$ is actually *Lipschitz*.

For the general situation, in [3, 38] the authors proved that shortest paths do have *Lipschitz continuous* first derivative, what means that in fact shortest paths are twice differentiable *almost everywhere* by Rademacher's Theorem. This fact will be of great importance below.

For a more comprehensive understanding of the theory of shortest paths and distance functions in Riemannian manifolds with boundary, see [2, 3, 38, 61] and references therein.

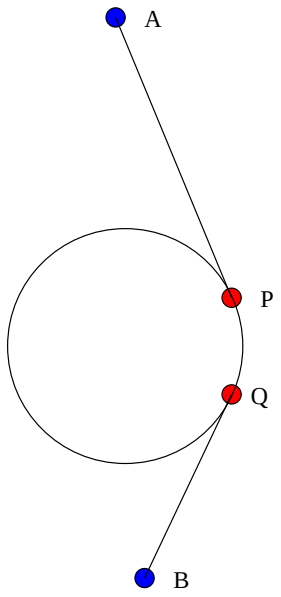


Figure 1: The minimal path is C^1, but not C^2.

2.3 Convergence Result for the Extrinsic Distance Function

We now show the relation between the Euclidean distance in the band Ω_h and the intrinsic distance in the surface $\mathcal{S}$. Below we will denote $d_{\mathcal{S}} \stackrel{\triangle}{=} d^1_{\mathcal{S}}$, and $d_{\Omega_h} \stackrel{\triangle}{=} d^1_{\Omega_h}$.

Observation 1 *Since we assume the implicit surface $\mathcal{S}$ to be compact, the continuous function $d_{\mathcal{S}} : \mathcal{S} \times \mathcal{S} \to I\!R$ attains its maximum. Therefore we can define the diameter of the set as*

$$diam(\mathcal{S}) \stackrel{\triangle}{=} \sup_{p,q \in \mathcal{S}} d_{\mathcal{S}}(p,q) < \infty$$

Observation 2 *Since $\mathcal{S} \subset \Omega_h$ we have that for every pair of points p and q in $\mathcal{S}$, $d_{\Omega_h}(p,q) \leq d_{\mathcal{S}}(p,q)$, so in view of the previous observation we have*

$$d_{\Omega_h}(p,q) \leq diam(\mathcal{S}) \;\; \forall\, p,q \in \mathcal{S}$$

Observation 3 *Since we are assuming $\tilde{g}$ to be a* smooth *extension of g to all $\Omega \supset \Omega_h$ (we stress the fact that the extension does not depend on h), $\tilde{g}$ will be Lipschitz in Ω, and we call $K_{\tilde{g}}$ its associated constant. Further, we will denote $M_g \stackrel{\triangle}{=} \max_{\{x \in \mathcal{S}\}} g(x)$ and $M_{\tilde{g}} \stackrel{\triangle}{=} \sup_{\{x \in \Omega\}} \tilde{g}(x)$.*

We need the following *Lemma* whose simple proof we omit (see for example [17]).

Lemma 1 *When a $\tilde{g}$-shortest path travels through an interior region, its curvature is absolutely bounded by*

$$B_{\tilde{g}} \stackrel{\triangle}{=} \sup_{\{x \in \Omega\}} \left(\frac{\|\nabla \tilde{g}(x)\|}{\tilde{g}(x)} \right)$$

The following Lemma will be needed in the proof of the Theorem below. Its proof can be found in Appendix B.

Lemma 2 *Let $f : [a,b] \to I\!R$ be a $C^1([a,b])$ function such that f' is Lipschitz. Let $\varphi \in L^\infty([a,b])$ denote (one of) f''s weak derivative(s). Then*

$$\int_a^b f'^2(x)\,dx = f\,f'|_a^b - \int_a^b f(x)\varphi(x)\,dx$$

We are now ready to present one of the main results of this Section. We bound the error between the intrinsic distance on $\mathcal{S}$ and the Euclidean one in the offset Ω_h. As we will see below, in the most general case, the error is of the order $h^{1/2}$ (h being half the offset width). We will later discuss that this is also the order of the theoretical error for the numerical approximation in fast marching methods. That will lead us to conclude that our algorithm does keep the convergence rate within the theoretically proven order for fast marching methods numerical approximation. However, for all practical purposes, the order of convergence in the numerical schemes used by fast marching methods is that of h, see [48]. We will also argue that for all practical purposes we can guarantee no decay in the overall rate of convergence. We defer the detailed discussion on this to after the presentation of the general bound below.

Theorem 2 *Let A and B be two points on the* smooth hypersurface *$\mathcal{S}$. Let $d_h^{\tilde{g}} = d_{\Omega_h}^{\tilde{g}}(A,B)$ and $d_{\mathcal{S}}^{g} = d_{\mathcal{S}}^{g}(A,B)$. Then we have that for sufficiently small h*

$$\left| d_{\mathcal{S}}^{g} - d_h^{\tilde{g}} \right| \leq h^{\frac{1}{2}}\; C(h)\; diam(S)$$

where $C(h)$ depends on the global curvature structure of $\mathcal{S}$ and on $\tilde{g}$, and approaches a constant when $h \downarrow 0$ (it does not depend on A nor B, we give a precise form of $C(h)$ in the proof).

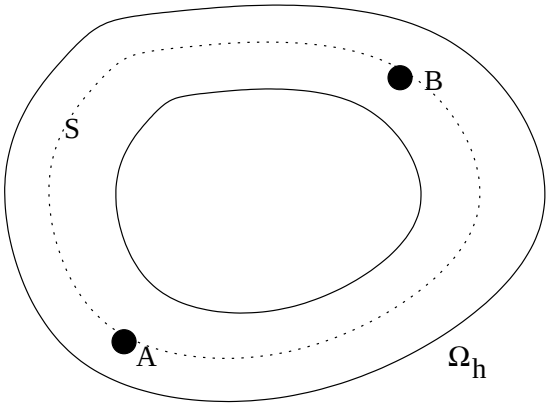


Figure 2: Tubular neighborhood.

Proof:
Let $d_h = d_{\Omega_h}(A,B)$; $d_{\mathcal{S}} = d_{\mathcal{S}}(A,B)$ and let $\gamma : [0, d_h]$ denote a Ω_h $\tilde{g}$-distance minimizing arc-length parameterized path between $A = \gamma(0)$ and $B = \gamma(d_h)$, such that $\|\dot{\gamma}\| = 1$. Let $\delta = \Pi_\psi(\gamma) = \gamma - \psi(\gamma)\nabla\psi(\gamma)$ be the orthogonal projection of γ onto $\mathcal{S}$. This will be as smooth as γ for small enough h, see Appendix A. For sufficiently small h, the boundary of Ω_h will be smooth, since $\mathcal{S}$ is smooth and no shocks *will be generated (see next Section and Appendix A). So we can assume that γ is C^1 and that $\dot{\gamma}$ is Lipschitz, since it is a* shortest path *within a smooth Riemannian manifold with boundary, see §2.2 above.*

It is evident that (this is a simple but key observation)

$$\mathbf{L}_{\tilde{g}}\{\gamma\} = d_h^{\tilde{g}} \overset{(1)}{\leq} d_{\mathcal{S}}^{g} \overset{(2)}{\leq} \mathbf{L}_g\{\delta\}$$

since

(1) $\mathcal{S} \subset \Omega_h$ *and* $\tilde{g}|_{\mathcal{S}} = g$

(2) δ *need not to be a* g*-shortest path between* A *and* B *on* $\mathcal{S}$.

We then have

$$\begin{aligned}
|\, d_{\mathcal{S}}^{g} - d_h^{\tilde{g}} \,| &\leq |\, \mathbf{L}_g\{\delta\} - \mathbf{L}_{\tilde{g}}\{\gamma\} \,| \;=\; |\, \mathbf{L}_{\tilde{g}}\{\delta\} - \mathbf{L}_{\tilde{g}}\{\gamma\} \,| \\
&\leq \int_0^{d_h} \Big|\; \tilde{g}(\delta)\|\dot{\delta}\| - \tilde{g}(\gamma)\|\dot{\gamma}\| \;\Big|\; dt \\
&\leq \int_0^{d_h} \Big|\tilde{g}(\delta)\|\dot{\delta}\| - \tilde{g}(\delta)\|\dot{\gamma}\|\Big|\; dt \;+\; \int_0^{d_h} |\, \tilde{g}(\delta)\|\dot{\gamma}\| - \tilde{g}(\gamma)\|\dot{\gamma}\| \,|\; dt \\
&= \int_0^{d_h} g(\delta) \Big|\|\dot{\delta}\| - \|\dot{\gamma}\|\Big|\; dt \;+\; \int_0^{d_h} |\tilde{g}(\delta) - \tilde{g}(\gamma)|\; dt \\
&\leq M_g \int_0^{d_h} \|\dot{\gamma} - \dot{\delta}\|\; dt \;+\; K_{\tilde{g}} \int_0^{d_h} \|\gamma - \delta\|\; dt \\
&= M_g \int_0^{d_h} \|\, \nabla\psi(\gamma)\cdot\dot{\gamma}\, \nabla\psi(\gamma) \;+\; \psi(\gamma)\, \mathbf{H}_{\psi}(\gamma)\dot{\gamma} \,\|\; dt \;+\; K_{\tilde{g}} \int_0^{d_h} \|\psi(\gamma)\nabla\psi(\gamma)\|\; dt \\
&\leq M_g \int_0^{d_h} |\, \nabla\psi(\gamma)\cdot\dot{\gamma} \,|\; dt \;+\; h\, M_g \int_0^{d_h} \|\, \mathbf{H}_{\psi}(\gamma)\dot{\gamma} \,\|\; dt \;+\; K_{\tilde{g}}\, h\, d_h
\end{aligned}$$

We now bound the first two terms at the end of the preceding expression.

1. *We first bound the second term in the preceding expression, this will be an ingredient to the bounding of the first term as well.*

   We have:

$$\|\, \mathbf{H}_{\psi}(\gamma)\dot{\gamma} \,\| \;\leq \sup_{\{v:\, \|v\|=1\;;\; p:\, d(p,\mathcal{S})\leq h\}} \|\, \mathbf{H}_{\psi}(p)v \,\| = \sup_{\{p:\, d(p,\mathcal{S})\leq h\}} \max\left(|\lambda(p)|,\, |\mu(p)|\right)$$

   where $\lambda(p)$ *and* $\mu(p)$ *denote the largest and the smallest eigenvalue of* $\mathbf{H}_{\psi}(p)$*, respectively.*

   Now, as we know from Appendix A the maximum absolute eigenvalue of $\mathbf{H}_{\psi}(p)$*,* $K(p)$*, is bounded by*

$$K(p) \leq \frac{\mathcal{M}_{\mathcal{S}}}{1 - |\psi(p)|\mathcal{M}_{\mathcal{S}}}$$

   where $\mathcal{M}_{\mathcal{S}}$ *is the maximum absolute eigenvalue of* $\mathbf{H}_{\psi}|_{\mathcal{S}}$*, that is*

$$\mathcal{M}_{\mathcal{S}} = \sup_{\{x\in\mathcal{S}\}} \max_{\{1\leq i\leq d\}} |\, \lambda_i\left(\mathbf{H}_{\psi}(x)\right) \,|$$

where $\lambda_i(\cdot)$ stands for the i-th eigenvalue of a symmetric matrix.

Then

$$\int_0^{d_h} \| \mathbf{H}_\psi(\gamma(s))\dot{\gamma}(s) \| ds \leq d_h \frac{\mathcal{M}_\mathcal{S}}{1 - h\mathcal{M}_\mathcal{S}}$$

2. *Let us define the function $f : [0, d_h] \to \mathbb{R}$, $f(t) = \psi(\gamma(t))$. Then formally $\dot{f}(t) = \nabla\psi(\gamma(t)) \cdot \dot{\gamma}(t)$ and $\ddot{f}(t) = \mathbf{H}_\psi(\gamma(t))[\dot{\gamma}(t), \dot{\gamma}(t)] + \nabla\psi(\gamma(t)) \cdot \ddot{\gamma}(t)$. Since $\dot{\gamma}(\cdot)$ is Lipschitz, and ψ is regular we can guarantee that $\dot{f}(\cdot)$ is also Lipschitz, so $\ddot{f}(\cdot)$ exists almost everywhere. We want to bound*

$$\int_0^{d_h} \left| \dot{f}(t) \right| \, dt$$

We note first that $f(0) = f(d_h) = 0$, and $|f(t)| \leq h$, $|\ddot{f}(t)| \leq \frac{\mathcal{M}_\mathcal{S}}{1-h\mathcal{M}_\mathcal{S}} + B_{\tilde{g}}$ for almost every $t \in [0, d_h]$. In fact, we have that for those sub-intervals of $[0, d_h]$ in which the shortest path travels through $\partial\Omega_h$, either $f(t) = h$, or $f(t) = -h$ for the whole subinterval, and therefore $f(t)$ is constant for each subinterval, so $\ddot{f}(t) = 0$ there. On the other hand, when γ is in the interior of Ω_h, it is a $\tilde{g}$-geodesic, so its acceleration is bounded by $B_{\tilde{g}}$, as we have seen in Lemma 1. Therefore, we conclude that $|\ddot{f}(t)| \leq |\mathbf{H}_{\psi(\gamma(t))}[\dot{\gamma}(t), \dot{\gamma}(t)]| + B_{\tilde{g}}$. Combining all this we have that for almost every $t \in [0, d_h]$,

$$|\ddot{f}(t)| - B_{\tilde{g}} \leq \sup_{\{\, v : \|v\| = 1;\, d(p,\mathcal{S}) \leq h\}} | \mathbf{H}_\psi(p)[v, v] | \leq \sup_{\{p :\, d(p,\mathcal{S}) \leq h\}} \max\left(|\lambda(p)|,\, |\mu(p)|\right)$$

and the given bound follows as before.

Applying Cauchy-Schwartz *inequality we obtain:*

$$\int_0^{d_h} | \dot{f}(t) | \, dt \leq \sqrt{(d_h) \int_0^{d_h} \dot{f}^2(t) \, dt}$$

Now using Lemma 2:

$$\int_0^{d_h} \dot{f}^2(t) \, dt = \dot{f} f \Big|_0^{d_h} - \int_0^{d_h} f \ddot{f} \, dt = - \int_0^{d_h} f \ddot{f} \, dt$$

$$\leq \int_0^{d_h} | f | \left| \ddot{f} \right| \, dt \leq (d_h)\, h \left(\frac{\mathcal{M}_\mathcal{S}}{1 - h\mathcal{M}_\mathcal{S}} + B_{\tilde{g}} \right)$$

Finally,

$$\int_0^{d_h} | \nabla\psi(\gamma) \cdot \dot{\gamma} | \, dt \leq (d_h) \sqrt{h \left(\frac{\mathcal{M}_\mathcal{S}}{1 - h\mathcal{M}_\mathcal{S}} + B_{\tilde{g}} \right)}$$

Using both computed bounds, we find that

$$| d_\mathcal{S}^g - d_h^{\tilde{g}} | \leq diam(\mathcal{S}) \sqrt{h} \left[M_g \sqrt{\frac{\mathcal{M}_\mathcal{S}}{1 - h\mathcal{M}_\mathcal{S}} + B_{\tilde{g}}} + M_g \sqrt{h} \frac{\mathcal{M}_\mathcal{S}}{1 - h\mathcal{M}_\mathcal{S}} + K_{\tilde{g}} \sqrt{h} \right] \tag{11}$$

■

From the preceding *Lemma* we obtain:

Corollary 1 *For a given point $q \in \mathcal{S}$*

$$d^{\tilde{g}}_{\Omega_h}\Big|_{\mathcal{S}}(q,\cdot) \overset{h\downarrow 0}{\rightrightarrows} d^{g}_{\mathcal{S}}(q,\cdot) \quad \text{in } \mathcal{S}$$

Remark 1 *The rate of convergence obtained with the techniques shown above is of order $\sqrt{h}$. A quick look over the proof of convergence shows that the term responsible for the $h^{1/2}$ rate is $\int_0^{d_h} \left| \dot{f}(t) \right| dt$. All other terms have the higher rate of h. Suppose we can find a* finite *collection of (disjoint) intervals $I_i = (a_i, b_i)$ such that $sgn(\dot{f})$ is constant (f is monotonic) within each I_i, $\cup_{i=1}^{N} I_i \subseteq [0, d_h]$ where N is the cardinality of that collection of intervals, and $\dot{f}(t) = 0$ for $t \in [0, d_h] \setminus \cup_{i=1}^{N} I_i$. Then, we could write:*

$$\begin{aligned}
\int_0^{d_h} \left| \dot{f}(t) \right| dt &= \sum_{i=1}^{N} sgn(\dot{f})\Big|_{(a_i,b_i)} \int_{a_i}^{b_i} \dot{f}(t)\, dt \\
&= \sum_{i=1}^{N} sgn(\dot{f})\Big|_{(a_i,b_i)} (f(b_i) - f(a_i)) = \sum_{i=1}^{N} |f(b_i) - f(a_i)| \\
&\leq \sum_{i=1}^{N} (|f(b_i)| + |f(a_i)|) \\
&\leq 2Nh \quad \textit{since} \quad f(t) = \psi(\gamma(t)) \ \textit{and} \ \gamma(\cdot) \ \textit{travels through} \ \Omega_h,
\end{aligned}$$

obtaining a higher rate of convergence, h. It is quite convincing that cases where $N = \infty$ can be considered pathological. We then argue that for all practical purposes the rate of convergence achieved is h. Notwithstanding, we are currently studying the space of surfaces and metrics g for which we can guarantee that $N < \infty$, and advances in this subject will be reported elsewhere.

This shows that we can approximate the intrinsic distance with the Euclidean one on the offset band Ω_h. Moreover, as we will detail below, the approximation error is of the same order as the theoretical numerical error in fast marching algorithms. Thereby, we can use fast algorithms in Cartesian grids to compute intrinsic distances (on implicit/implicitized surfaces), enjoying their computational complexity without affecting the convergence rate given by the underlying numerical approximation scheme.

3 Numerical Implementation and its Theoretical Error

In this section we first discuss the simple modification that needs to be incorporated into the (Cartesian) fast marching algorithm in order to deal with Euclidean spaces with manifolds. We then propose a way of estimating the (now discrete) offset h, and bound the total numerical error of our algorithm, thereby showing our assertion that the error with our algorithm is of the same order than the one obtained with the fast marching algorithm for Cartesian grids (or triangulated 3D surfaces).

As stated before, we are dealing with the numerical implementation of the Eikonal equation inside an open, bounded and connected domain Ω (this will later become the offset Ω_h). The general equation, when $P(x)$ is the weight (it becomes $\tilde{g}$ for our particular case), is given by

$$\begin{cases} \|\nabla f(x)\| = P(x) \quad \forall\, x \in \Omega \\ f(r) = 0 \end{cases} \tag{12}$$

being r the seed point. Note that following the results in the previous section, we are now dealing with the Eikonal equation in Euclidean space, and then the Euclidean gradient is used above.

The upwind numerical scheme to be used for this equation is of the form ($\Delta x_1 = \Delta x_2 = \ldots = \Delta x_d \;=\; \Delta x$) [48]:

$$\begin{cases} \sum_{j=1}^{d} max^2(\widehat{f}(p) - m_j, 0) = (\Delta x)^2\, P^2(p) \\ m_j = min(\widehat{f}(p + \Delta x \vec{e}_j), \widehat{f}(p - \Delta x \vec{e}_j)) \end{cases} \tag{13}$$

where $\widehat{f}$ is the numerically computed value of f for every point p in the discrete domain

$$\mathcal{D}(\Omega, \Delta x) \stackrel{\triangle}{=} \Omega \cap (\mathbb{Z}\Delta x)$$

Here, $\vec{e}_j$ with $j = 1, 2, ..., d$, are the elements of the canonical basis of $\mathbb{R}^d$.

We now describe the fast marching algorithm for solving the above equation. For this we follow the presentation in [51]. For clarity we write down the algorithm in *pseudo-code* form. Details on the original fast marching method on Cartesian grids can be found in the mentioned references.

At all times there are 3 kinds of points under consideration:

- **NarrowBand**. These points have to them associated an already guessed value for $\widehat{f}$, and are immediate neighbors to those points whose value has already been "frozen."
- **Alive**. These are the points whose $\widehat{f}$ value has already been frozen.
- **Far Away**. These are points that haven't been processed yet, so no tentative value has been associated to them. For that reason they have $\widehat{f} = \infty$.

The steps of the algorithm are:

- *Initialization*:
  1. Set $\widehat{f} = 0$ for every point belonging to the set **Alive** (these are the seed/s point/s).
  2. Find a tentative value of $\widehat{f}$ for every **Neighbor** of an **Alive** point and tag them **NarrowBand**.
  3. Set $\widehat{f} = \infty$ for all the remaining points in the discrete domain.
- *Advance*:
  1. Beginning of loop: Let (p_{min}) be the point $\in$ [**NarrowBand**] which takes the least value of $\widehat{f}$.
  2. Insert the point p_{min} to the set [**Alive**] and remove it from [**NarrowBand**].

3. Tag as **Neighbors** all those points in the discrete domain that can be written in the form $p_{min} \pm \Delta x\, \vec{e}_j$, and belonging to [**NarrowBand**] $\cup$ [**FarAway**]. If a **Neighbor** is in [**FarAway**], remove it from that set and insert it to [**NarrowBand**].
4. Recalculate $\widehat{f}$ for all **Neighbors** using equation (13)
5. Back to the beginning.

The boundary conditions are taken such that points beyond the discrete domain have $\widehat{f} = \infty$.

The condition that is checked all the time, and that really defines the domain the algorithm is working within, is the one that determines if a certain point q is **Neighbor** of a given point p that belongs to the domain. The only thing one has to do in order to make the algorithm work in the domain Ω_h specified by $\{x \in I\!R^d : |\psi(x)| \leq h\}$ is change the way the **Neighbor** checking is done. More precisely, we should check

$$q \in \textbf{Neighbor}(p) \text{ iff } \{(|\psi(q)| \leq h) \;\&\&\; (q \text{ can be written like } p \pm \Delta x\, \vec{e}_j\ )\}$$

the emphasis here being on the test "$|\psi(q)| \leq h$." We could also achieve the same effect by giving an infinite weight to all points outside Ω_h, that is, we treat the outside of Ω_h as an obstacle. That is, with an extremely simple modification to the fast marching algorithm, we make it work as well for distances on manifolds with boundary, and therefore, for intrinsic distances on implicit surfaces. This is of course supported by the convergence results in the previous section and the analysis on the numerical error presented below.

3.1 Bounding the Offset h

We now present a technique to estimate h, the size of the offset of the hypersurface $\mathcal{S}$ that actually defines the computational domain Ω_h. The bounds on h are very simple. On one hand, we need h to be large enough so that the upwind scheme can be implemented, meaning that h has to be large enough to include the stencil used in the numerical implementation. On the other hand, h has to be small enough to guarantee that Ω_h remains simply connected with smooth boundaries and that $\tilde{g}$ remains smooth inside Ω_h.

Let $\mathcal{M}_{\mathcal{S}}$ be as before a bound for the absolute sectional curvature of $\mathcal{S}$, and let Δx be the grid size. In addition, let W be the maximal offsetting of the surface $\mathcal{S}$ that guarantees that the resulting set remains connected and different parts of the boundary of that set do not touch each other. We show below that a suitable bounding of h is

$$\Delta x \sqrt{d} \;<\; h \;<\; \min\left\{\frac{1}{\mathcal{M}_{\mathcal{S}}}, W\right\}. \tag{14}$$

Let us introduce some additional notation. We denote by *cell* to the unit cell of the computational grid. Let x be a point in Ω_h, we denote by $n(x)$ the number of cells $C_1(x), \ldots, C_{n(x)}$ that contain x. It is clear that if $x \in \mathcal{D}(\Omega_h, \Delta x)$ (it is a grid point), then x is contained in 2^d cells having x as a vertex. It is also clear that $n(x) \leq 2^d$. For a given cell C we call $P(C)$ the set of points of $\mathcal{D}(\Omega_h, \Delta x)$ that compose C (i.e., its vertices). We will denote by $\mathbf{C}(x)$ the set $\bigcup_{i=1}^{n(x)} C_i(x)$, and by $\mathbf{P}(x)$ the set $\left(\bigcup_{i=1}^{n(x)} P(C_i(x))\right)^*$ where the "*" means that we remove repeated elements (points).

The lower bound comes from forcing that for every $x \in \mathcal{S}$, all points in $\mathbf{C}(x)$ lie within Ω_h (note of course that we want h as small as possible). That is:

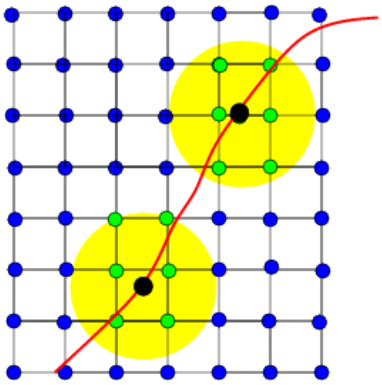


Figure 3: We depict the situation that leads to the lower bound for h in the $2D$ case. In red: the curve. In black: the centers of $\overline{B}(x \in \mathcal{S}, d^{1/2}\Delta x)$. In green: the points of $\mathcal{D}(\Omega_h, \Delta x)$ that fall inside $\overline{B}(x, d^{1/2}\Delta x)$ for some $x \in \mathcal{S}$, and in blue those that don't.

$$\bigcup_{x \in \mathcal{S}} \mathbf{C}(x) \subset \Omega_h$$

Once again, this constraint comes in order to guarantee that there are "enough" points to make the discrete calculations. We try to make $\mathbf{C}(x) \subset \overline{C}(x, l)$, where $C(x, l)$ stands for the hypercube centered in x, with side length $2l$, and sides parallel to the gridding directions. The worst scenario is when x is a point in the discrete domain, and we must have $l \geq \Delta x$. Finally, we observe that $C(x, l) \subset B(x, l\sqrt{d})$. The condition then becomes

$$\bigcup_{x \in \mathcal{S}} B(x, \Delta x\sqrt{d}) \subset \Omega_h = \bigcup_{x \in \mathcal{S}} B(x, h)$$

which provides the lower bound, $h > \Delta x\sqrt{d}$.

The upper bound includes two parts. First, we shouldn't go beyond W, since if we do so, different parts of the offset surface might touch each other, situation which can even create a non-simple band Ω_h. The second part of the upper bound comes from seeking that when traveling on a characteristic line of ψ at a point p of $\mathcal{S}$, no shocks occur inside Ω_h. It is a simple fact that this won't happen if $h < \frac{1}{\mathcal{M}_{\mathcal{S}}}$, see Appendix A. It is extremely important to guarantee this both to obtain smooth boundaries for Ω_h and to obtain smooth extensions of the metric g ($\tilde{g}$).

Note of course that in general, h and also Δx can be position dependent. We can use an adaptive grid, and in places where the curvature of $\mathcal{S}$ is high, or places where high accuracy is desired, we can make Δx small.

3.2 The Numerical Error

It is time now to explicitly bound the numerical error of our proposed method. As stated above, it is our goal to formally show that we are within the same order that the computationally optimal (fast marching) algorithms for computing distance functions on Cartesian grids. Note that the numerical error for the fast marching algorithm on triangulated surfaces has not been reported, although it is of course bounded by the Cartesian one (since this provides a particular "triangulation").

3.2.1 Numerical Error Bound of the Cartesian Fast Marching

The aim of this section is to bound a quantity that measures the difference between the numerically computed value $\widehat{d^g_{\mathcal{S}}}(p,\cdot)$ and the real value $d^g_{\mathcal{S}}(p,\cdot)$. Any such quantity will be comparing both functions on $\mathcal{S}$, but in principle the numerically computed value will not be defined all over the hypersurface. So we will be dealing with an interpolation stage, that we comment further below in §3.2.2.

Let us fix a point $p \in \mathcal{S}$, and let $\widehat{f}(\cdot)$ be the numerically computed solution (according to (13)), and $f(\cdot)$ the *real* viscosity solution of the problem (12). The approximation error is then bounded by (see [48])

$$\max_{p\in\mathcal{D}(\Omega,\Delta x)} |\widehat{f}(p) - f(p)| \leq C_L(\Delta x)^{\frac{1}{2}} \tag{15}$$

where C_L is a constant. In practice, however, the authors of [48] observed first order accuracy. As we have seen, we have also find an error of order $h^{1/2}$ for the general approximation of the weighted intrinsic distance on $\mathcal{S}$ with the distance in the band Ω_h, and a practical order of h (see Remark 1 and Theorem 2).

Before proceeding with the presentation of the whole numerical error of our proposed algorithm, we need the following simple Lemma whose proof we omit.

Lemma 3 *For a convex set $D \subset \Omega$, and $y, z \in D$, f satisfies*

$$|f(z) - f(y)| \leq \|P\|_{L^\infty(\Omega)}\|z - y\|$$

Remark 2 *Using the preceding Lemma and (15), it is easy to see that for x such that $\mathbf{C}(x) \subset \Omega$:*

$$|\widehat{f}(p) - \widehat{f}(q)| \leq 2C_L(\Delta x)^{\frac{1}{2}} + \|P\|_{L^\infty(\Omega)}\sqrt{d}\Delta x, \quad \forall p, q \in \mathbf{P}(x) \tag{16}$$

a relation we will shortly use.

3.2.2 The Interpolation Error

Since following our approach we are now computing the distance function in the band Ω_h, in the corresponding discrete Cartesian grid, we have to interpolate this to obtain the distance on the zero level-set $\mathcal{S}$. This interpolation produces a numerical error which we now proceed to bound.

Given the function $\zeta : \mathcal{D}(\Omega, \Delta x) \to I\!R$, we define the function $\mathcal{I}(\zeta) : \Omega \to I\!R$ through an interpolation scheme. We will assume that the interpolation error is bounded in the following way:[8]

$$\sup_{y\in\mathbf{P}(x)} |\zeta(y) - \mathcal{I}(\zeta)(x)| \leq \max_{z\in\mathbf{P}(x)} \zeta(z) - \min_{z\in\mathbf{P}(x)} \zeta(z)$$

for every $x \in \Omega$

[8] One may imagine several interpolation schemes satisfying this not-stringent-at-all condition.

3.2.3 The Total Error

We now present the complete error (numerical plus interpolation) introduced by our algorithm, without considering the possible error in the computation of $\tilde{g}$ (or in other words, we assume that the weight was already given in the whole band Ω_h).

Let p be a point in $\mathcal{S}$. We denote by

- $d^g_{\mathcal{S}}(p,\cdot) : \mathcal{S} \to I\!R$ the *intrinsic* g-distance function from p to any point in $\mathcal{S}$.
- $d^{\tilde{g}}_h(p,\cdot) : \Omega_h \to I\!R$ the $\tilde{g}$-distance function from p to any other point in Ω_h.
- $\widehat{d^{\tilde{g}}_h}(p,\cdot) : \mathcal{D}(\Omega_h, \Delta x) \to I\!R$ the *numerically computed* value of $d^{\tilde{g}}_h(p,\cdot)$ to any point in the discrete domain.
- $\mathcal{I}\left(\widehat{d^{\tilde{g}}_h}\right)(p,\cdot) : \mathcal{S} \to I\!R$ the result of interpolating $\widehat{d^{\tilde{g}}_h}$ (that's only specified for points in $\mathcal{D}(\Omega_h, \Delta x)$) to points in $I\!R^d \supset \mathcal{S}$.

The goal is then to bound $\left\| d^g_{\mathcal{S}}(p,\cdot) - \mathcal{I}\left(\widehat{d^{\tilde{g}}_h}\right)(p,\cdot) \right\|_{L^\infty(\mathcal{S})}$, and we proceed to do so now.

Let x be in $\mathcal{S}$ and y in $\mathbf{P}(x)$, then:

$$\left| d^g_{\mathcal{S}}(p,x) - \mathcal{I}\left(\widehat{d^{\tilde{g}}_h}\right)(p,x) \right| \le |d^g_{\mathcal{S}}(p,x) - d^{\tilde{g}}_h(p,x)| + |d^{\tilde{g}}_h(p,x) - d^{\tilde{g}}_h(p,y)| \tag{17}$$
$$+ \left| d^{\tilde{g}}_h(p,y) - \widehat{d^{\tilde{g}}_h}(p,y) \right| + \left| \widehat{d^{\tilde{g}}_h}(p,y) - \mathcal{I}\left(\widehat{d^{\tilde{g}}_h}\right)(p,x) \right|$$

and using Proposition 2, Lemma 3, (15) and simple manipulations (in that order) we obtain:

$$\left| d^g_{\mathcal{S}}(p,x) - \mathcal{I}\left(\widehat{d^{\tilde{g}}_h}\right)(p,x) \right| \le C(h)\, diam(\mathcal{S})\, h^{\frac{1}{2}} + M_{\tilde{g}}\|x-y\| + C_L(\Delta x)^{\frac{1}{2}}$$
$$+ \left(\max_{y \in \mathbf{P}(x)} \widehat{d^{\tilde{g}}_h}(p,y) - \min_{y \in \mathbf{P}(x)} \widehat{d^{\tilde{g}}_h}(p,y) \right)$$

The last term can be dealt with using (16). Taking into account that we will always choose $h = C_x^2\, \Delta x \sqrt{d}$ for some constant $C_x > 1$ (as we saw in §3.1), we conclude:

$$\left\| d^g_{\mathcal{S}}(p,\cdot) - \mathcal{I}\left(\widehat{d^{\tilde{g}}_h}\right)(p,\cdot) \right\|_{L^\infty(\mathcal{S})} \le (\Delta x)^{\frac{1}{2}}\, \mathcal{C}(\Delta x; \mathcal{S}) \tag{18}$$

where $\mathcal{C}(\Delta x; \mathcal{S})$ goes to a constant (that depends on $\mathcal{S}$) as $\Delta x \downarrow 0$, and this provides the desired bound. Note that as announced below, the error of our algorithm is of the same (theoretical) order as the error of the fast marching on Cartesian grids, thereby justifying our approach.

We have then obtained that the use of an Euclidean approximation in the band Ω_h to the intrinsic distance function on the level-set $\mathcal{S}$ doesn't change the order of the whole numerical approximation. In the most general case, the theoretical bound is of order $h^{1/2}$, while for all practical purposes is of order h (when we replace in the computation above the practical bounds for the fast marching algorithm and for the distance approximation).

To conclude, let's point out that since we are working within a narrow band (Ω_h) of the surface $\mathcal{S}$, we are actually not increasing the dimensionality of the problem. We can then work with a Cartesian grid while keeping the same dimensionality as if we were working on the surface.

4 Experiments

We now present a number of 3D examples of our algorithm. Recall that although all the examples are given in 3D, the theory presented above is valid for any dimension $d \geq 3$.

Two classes of experimental results are presented. We show a number of intrinsic distance functions for implicit surfaces, as well as geodesics computed using these functions. In order to compute interesting geodesics, we use also non-uniform weights, permitting the computation of crest/valleys, and optimal paths with obstacles. We also experimentally compare our results with those obtained on triangulated surfaces using the fast marching technique developed by Kimmel and Sethian (for this we use the results and software reported in [6]).

Figures 4, 5, and 6 show the intrinsic distance function for implicit surfaces computed with the method here proposed ($g = 1$). An arbitrary seed point on the implicit surface has been chosen, and pseudo colors are used to improve the visualization. Red corresponds to low values of the distance and blue to the high ones. We observe that, as expected, the distance (colors) vary smoothly, and that close points have similar colors and far points have very different colors (close and far measured on the surface of course).

In Figure 7 we compare the result of our approach with that of fast marching on a triangulated surface (this later computed with the package reported in [6]), while in Figure 8 we show level lines of the intrinsic distance function computed with the technique here proposed.

Before concluding this part of the experiments, let's give some technical details on the implementation. The code for the examples in this paper was written in `C++`. For visualization purposes, VTK was used. Most of the "hard code" was done taking advantage of Blitz++'s double templatized arrays and related routines, see [9]. The implicit models used in this paper were obtained from [65] (other techniques, e.g., [39], could be used as well). All the code was compiled and run in a $450\,Mhz$ Pentium *III*, with $256\,Mb$ of RAM, working under Linux (RedHat 6.2). The compiler used was `egcs-2.91.66` and the level of optimization was 3. In the table below we show running times of the intrinsic distance map algorithm for some of the implicit models we used, along with the corresponding *offset*-value (h) and size and number of grid points in Ω_h for each model.

Model	**Size**	$\#\mathcal{D}(\Omega_h, \Delta x)$	**h**	**Running Time** (secs)
Brain	$122 \times 142 \times 124$	$168,603$	1.75	9.4
Bunny	$81 \times 80 \times 65$	$38,107$	1.75	1.99
Knot	$80 \times 81 \times 44$	$16,095$	1.0	0.76
Sphere	$70 \times 70 \times 70$	$11,800$	1.75	0.65
Torus	$64 \times 64 \times 64$	$21,704$	1.75	1.16
Teapot	$80 \times 55 \times 46$	$24,325$	1.75	1.22

4.1 Geodesics on Implicit Surfaces

To find geodesic curves on the implicit surface, we back track starting from a specified target point toward the seed point, while traveling on the surface in the direction given by the (negative) intrinsic-distance gradient. This means that after we have computed the intrinsic distance function as explained above, we have to solve the following **ODE** (which obviously keeps the curve on $\mathcal{S}$):

$$\begin{cases} \dot{\gamma} = -\nabla_{\psi} d^{\tilde{g}}_{\Omega_h}(\gamma) \\ \gamma(0) = p \in \mathcal{S} \end{cases}$$

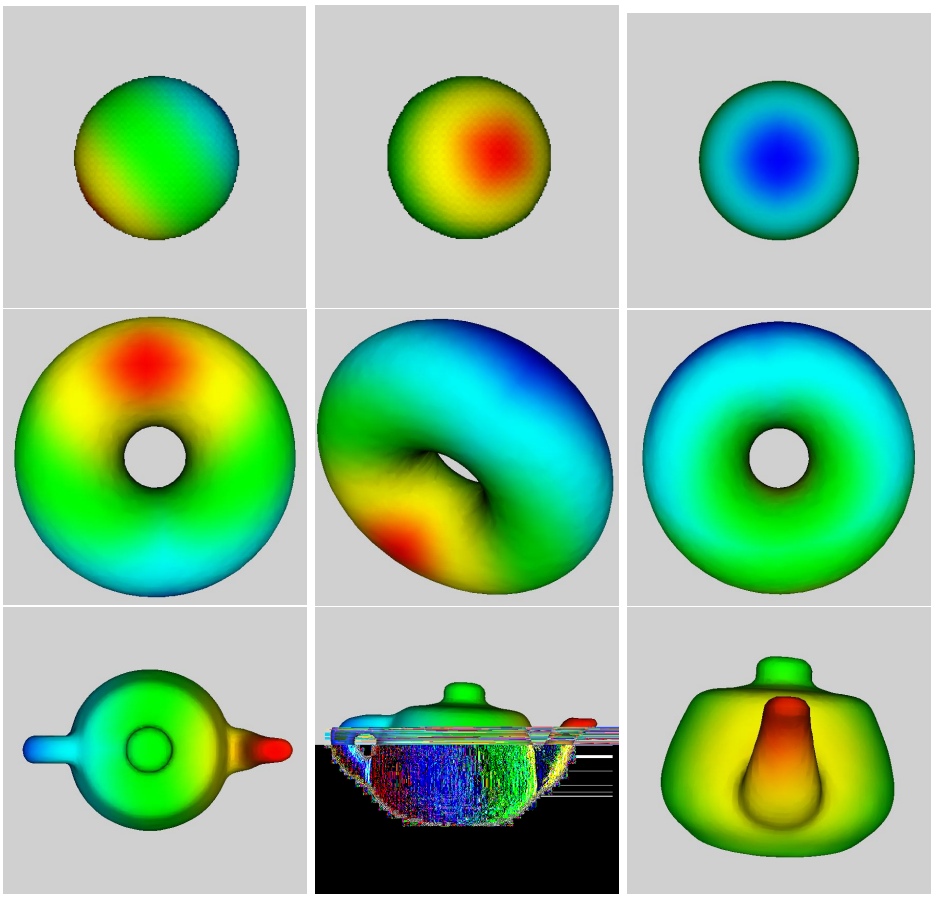


Figure 4: Distance maps from a point on the sphere, torus and teapot (three views are presented for each model).

where $\nabla_\psi d^{\tilde{g}}_{\Omega_h}(p) \triangleq \nabla d^{\tilde{g}}_{\Omega_h}(p) - \left(\nabla d^{\tilde{g}}_{\Omega_h}(p) \cdot \nabla\psi(p)\right) \nabla\psi(p)$ is the gradient of $d^{\tilde{g}}_{\Omega_h}$ at $p \in \mathcal{S}$ projected onto the tangent space to $\mathcal{S} = \{\psi = 0\}$ at p. Since we must discretize the above equation, one can no longer assume that at every instant the geodesic path γ will lie on the surface, so a projection step must be added. In addition, since all quantities are known only at grid points, an interpolation scheme must be used to perform all evaluations at positions given by γ. We have used a simple Runge-Kutta integration procedure, with adaptive step, namely an ODE23 procedure.

Before presenting examples of geodesic curves, we should note that we are assuming that $\nabla_\psi d^{\tilde{g}}_{\Omega_h}$ is a good approximation of $\nabla_{\mathcal{S}} d^g_{\mathcal{S}}$ (and not just $d^{\tilde{g}}_{\Omega_h}$ a good approximation of $d^g_{\mathcal{S}}$ as we have previously proved). Bounding the error between these two gradients, e.g., using the framework of viscosity solutions, is the subject of current work.

The figures described next illustrate the computation of geodesic curves on implicit surfaces for different weights g. In all the figures the geodesic curve is drawn on top of the surface, which is colored as before, colors indicating the intrinsic weighted distance.

In Figure 9 we present both the geodesic curve computed with our technique and the one computed with the fast marching algorithm on triangulated surfaces following the implementation reported in [6].

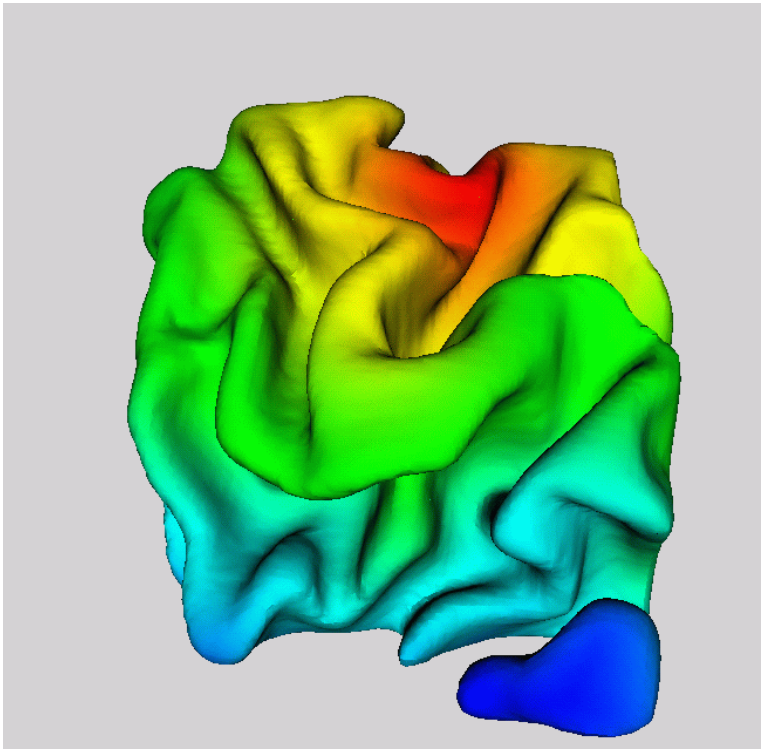


Figure 5: Distance map from a point on a portion of white/gray matter boundary of the cortex.

In Figure 10 we show the computation of sulci (valleys) on an implicit surface representing the boundary between the white and gray matter in a portion of the human cortex (data obtained from MRI). Here the (extended) weight $\tilde{g}$ is a function of the mean curvature given by [6]

$$\tilde{g}_{valley}(x) = \omega + \left(\mathbf{M}(x) - \min_{y \in \Omega_h} \mathbf{M}(y) \right)^p$$

where $\mathbf{M}$ stands for the mean curvature of the level sets of ψ, so it is computed simply as $\mathbf{M}(x) = \Delta\psi(x)$. In the example presented we used $\omega = 100$ and $p = 3$. More details on the use of this approach for detecting valleys (and creases) can be found in [6] and in the references therein.

In Figure 11 we show the computation of geodesic curves with obstacles on implicit surfaces. This is an important computation for topics such as motion planning on surfaces.

5 Extensions

5.1 General Metrics: Solving Hamilton-Jacobi Equations on Implicit Surfaces

Since the very beginning of our exposition we have restricted ourselves to *isotropic* metrics. As stated in the introduction, this already has a tremendous amount of applications, and just a few were shown in the previous section. Since the fast marching approach has been recently extended to more general Hamilton-Jacobi equations by Osher and Helmsen [44], we are immediately tempted to extend our framework to these equations as well (these equations have applications in important areas such as adaptive mesh generation on manifolds, [27], and semiconductors manufacturing).

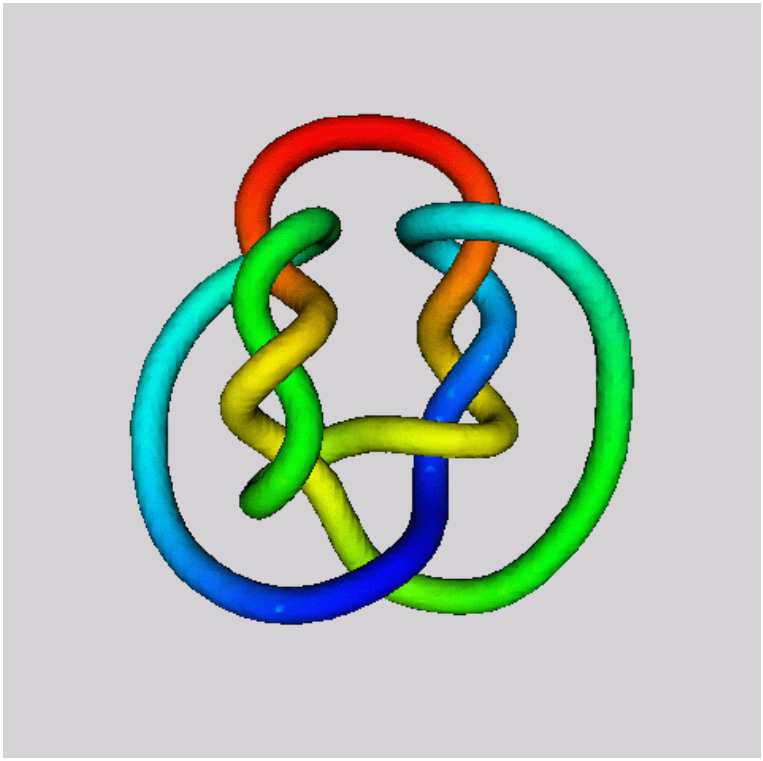


Figure 6: Distance map from one seed point on a knot. In this picture we evidence that the algorithm works well for quite convoluted geometries (as long as h is properly chosen). Note how points close in the Euclidean sense but far away in the intrinsic sense receive very different colors, indicating their large (intrinsic) distance.

Then, we are led to investigate the extension of our algorithm to general metrics of the form, $\mathbf{G} : \mathcal{S} \to I\!R^{d \times d}$, that is, a positive definite 2-tensor. Our new definition of weighted length becomes

$$\mathbf{L}_{\mathbf{G}}\{\mathcal{C}\} \stackrel{\triangle}{=} \int_a^b \sqrt{\mathbf{G}(\mathcal{C}(t))[\dot{\mathcal{C}}(t), \dot{\mathcal{C}}(t)]}\ dt$$

and the problem is to find for every $x \in \mathcal{S}$ (for a fixed $p \in \mathcal{S}$)

$$d_{\mathcal{S}}^{\mathbf{G}}(x, p) \stackrel{\triangle}{=} \inf_{\mathcal{C}_{px}[\mathcal{S}]} \{\mathbf{L}_{\mathbf{G}}(\mathcal{C})\} \tag{19}$$

As before, we attempt to solve the approximate problem in the band Ω_h, with an extrinsic distance:

$$d_{\Omega_h}^{\tilde{\mathbf{G}}}(x, p) \stackrel{\triangle}{=} \inf_{\mathcal{C}_{px}[\Omega_h]} \{\mathbf{L}_{\tilde{\mathbf{G}}}(\mathcal{C})\} \tag{20}$$

where

$$\mathbf{L}_{\tilde{\mathbf{G}}}\{\mathcal{C}\} \stackrel{\triangle}{=} \int_a^b \sqrt{\tilde{\mathbf{G}}(\mathcal{C}(t))[\dot{\mathcal{C}}(t), \dot{\mathcal{C}}(t)]}\ dt$$

for an adequate extension $\tilde{\mathbf{G}}$ of $\mathbf{G}$. The solution of the extrinsic problem satisfies (in the viscosity sense) the Eikonal equation

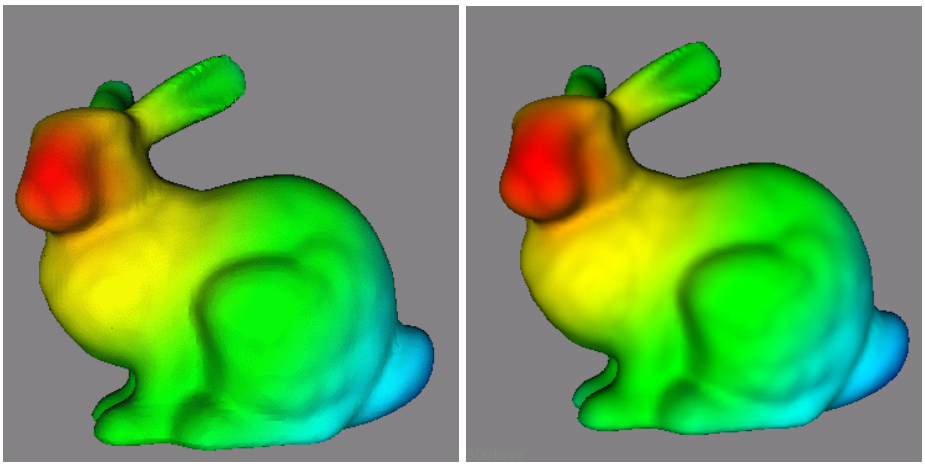


Figure 7: Distance map from a single seed point (situated at the nose) on an implicit bunny ($g = 1$). The figure on the left was obtained with the implicit approach here presented, while the one on the right was derived with the fast marching on triangulated surfaces technique.

$$(\tilde{\mathbf{G}}^{-1})(x)[\nabla d^{\tilde{\mathbf{G}}}_{\Omega_h}, \nabla d^{\tilde{\mathbf{G}}}_{\Omega_h}] = 1 \tag{21}$$

The first issue now is the numerical solvability of the preceding equation using a fast marching type of approach. Osher and Helmsen, [44], have extended the capabilities of the fast marching to deal with Hamilton-Jacobi equations of the form

$$H(x, \nabla f) = a(x)$$

for geometrically based Hamiltonians $H(x, \vec{p}) : \Omega(\subset I\!R^d) \times I\!R^d \to I\!R$ that satisfy

$$\begin{cases} H(x, \vec{p}) > 0 \text{ if } \vec{p} \neq \vec{0} \\ H(x, \vec{p}) \text{ is homogeneous of degree 1 in } \vec{p} \end{cases} \tag{22}$$

We easily observe that these conditions hold for (21) considering $H(x, \vec{p}) \stackrel{\triangle}{=} \sqrt{(\tilde{\mathbf{G}}^{-1})(x)[\vec{p}, \vec{p}]}$, and therefore we can solve these type of Hamilton-Jacobi equations (the extrinsic problem) with the extended fast marching algorithm.

In order to show that our framework is valid for these equations as well, all what we basically need to do is to prove that the extrinsic distance (20) on the offset Ω_h converges to the intrinsic one on the implicit surface $\mathcal{S}$, i.e., (19). This can be done repeating the steps in the convergence proof previously reported in §2.3 for isotropic metrics. Combining this with the results of Osher and Helmsen we then obtain that our framework can be applied to a larger class of Hamilton-Jacobi equations.

5.2 Non Implicit Surfaces

The framework we presented was here developed for implicit surfaces, although it applies to other surface representations as well. First, if the surface is originally given in polygonal or triangulated form, or even as a set of unconnected points and curves, we can use a number of available techniques, e.g., [33, 39, 46, 54, 63, 65] (and some very nice public domain software [39]), to first implicitize

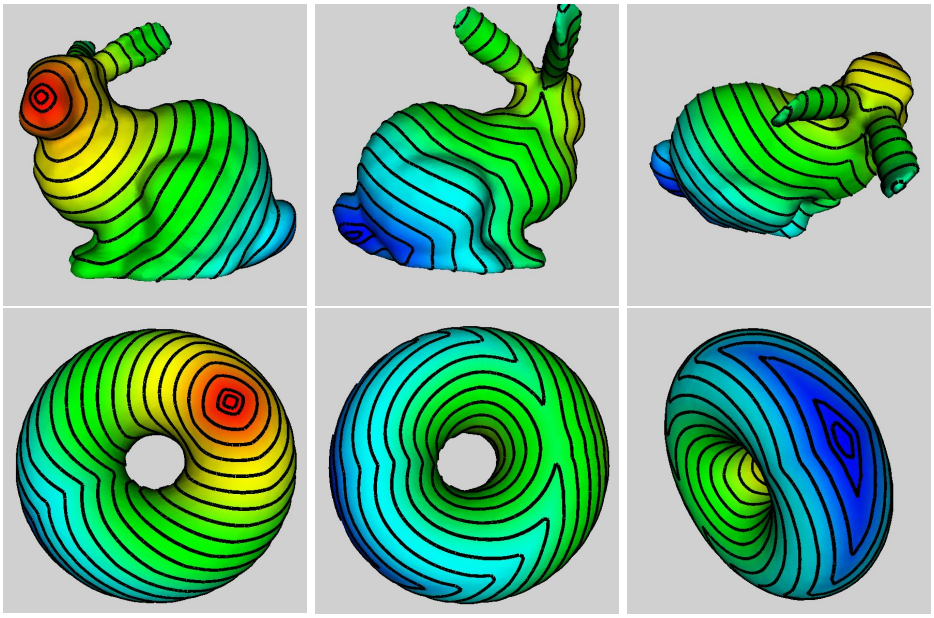


Figure 8: Top: Level lines for the intrinsic distance function depicted in Figure 7 (left). Bottom: Level lines for the intrinsic distance function depicted in Figure 4 (second row). In both rows, the (22) levels shown are 0.03, 0.05, 0.1, ..., 0.95, 0.97 percent of the maximum value of the intrinsic distance, and the coloring of the surface corresponds to the intrinsic distance function. Three views are presented. Note the correct separation between adjacent level lines. Note also how these lines are "parallel".

the surface and then apply the technique here proposed.[9] Note that the implicitation needs to be done only once per surface as a pre-processing step, and will remain valid for all subsequent uses of the surface. Moreover, as we have seen, all what we need is to have a Cartesian grid in a small band around the surface $\mathcal{S}$. Therefore, there is no explicit need to perform an implicitation of the given surface representation. For example, if the surface is given by a cloud of unconnected points, we can compute distances intrinsic to the surface defined by this cloud, as well as intrinsic geodesic curves, without explicitly computing the underlying surface. All what is needed is to embed this cloud of points in a Cartesian grid and consider only those points in the grid at a distance h or less from the points in the cloud. The computations are then done on this band.

6 Concluding Remarks

In this article we have presented a novel computationally optimal algorithm for the computation of intrinsic distance functions and geodesics on implicit hyper-surfaces. The underlying idea is based on using the classical Cartesian fast marching algorithm in an offset bound around the given surface. We have provided theoretical results justifying this approach and presented a number of experimental examples. The technique can also be applied to 3D triangulated surfaces, or even surfaces represented by clouds of unconnected points, after these have been embedded in a Cartesian grid with proper boundaries. The conclusion is that there is no need to further develop algorithms for intrinsic distances and geodesics on hyper-surfaces, original Cartesian approaches are sufficient. We have also discussed that the approach is valid for more general Hamilton-Jacobi equations as well.

[9] The same techniques can be applied to transform any given implicit function into a distance one.

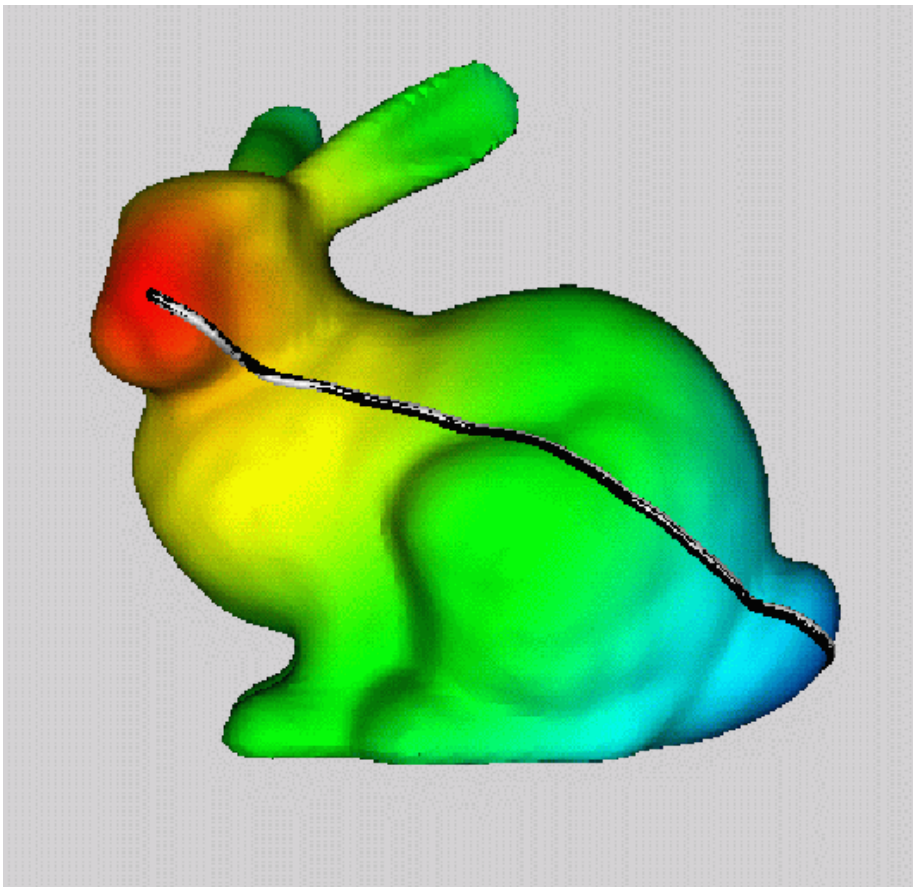


Figure 9: Distance map and geodesic curve between two points on an implicit bunny. We show two geodesics superimposed, the black one is the one obtained via the implicit back propagation described in the text, while the white one is obtained when performing the back propagation computation in the triangulated surface. It is important to note that in both cases the distance function used is the one computed with our implicit approach; to feed the data to the triangulated surfaces back-propagation algorithm, we first interpolated the intrinsic distance to points onto the triangulated surface. We can clearly see that both geodesics overlap almost entirely.

Many questions remain open. First, we are currently working on tighter bounds for the error between $d^{\tilde{g}}_{\Omega_h}$ and $d^{g}_{\mathcal{S}}$, as well as bounds for the error between their corresponding derivatives. We are interested in extending the framework here presented to the computation of distance functions on high codimension surfaces and general embeddings. More generally, it remains to be seen what class of intrinsic Hamilton-Jacobi (or in general, what class of intrinsic PDE's) can be approximated with equations in the offset band Ω_h. In an even more general approach, what kind of intrinsic equations can be approximated by equations in other domains, being offsets just a particular and important example. Even if fast marching techniques do not exists for these equations, it might be simpler and even more accurate to solve the approximating equations in these domains than in the original surface $\mathcal{S}$. The framework here presented then not only offers a solution to a fundamental problem, but it also opens the doors to a new area of research.

Acknowledgments

Many colleagues and friends helped us during the course of the work here reported, and they deserve our most sincere acknowledgment. Prof. Miguel Paternain, in the early stages of this work, pointed out very relevant references. Prof. Stephanie Alexander, Prof. David Berg, and Prof. Richard Bishop helped us with our questions about geodesics on manifolds with boundaries. Prof. Stanley Osher helped us with issues regarding the fast marching algorithm in general, and his recent extensions in particular, while Prof. Ron Kimmel discussed with us topics related to his triangulated work and provided general comments on the paper. Prof. Osher also helped with the literature on rate of convergence of Hamilton-Jacobi numerical approximations. Prof. Omar Gil helped us with some deep insights into technical aspects of the work. This work benefited from rich conversations with Alberto Bartesaghi (who also provided us with some of his code to extract geodesics in triangulated surfaces), Cesar Perciante, Luis Vásquez, Federico Lecumberry, and Prof. Gregory Randall. We also enjoyed important discussion with Prof. Vicent Caselles on the density of zeros of general functions. We also thank Prof. Luigi Ambrosio for his comments on the regularity of level-sets of distance functions. We are indebted to Alvaro Pardo for his careful reading of the manuscript and for the suggestions he made. Prof. H. Zhao provided some of the implicit surfaces. FM in particular wants to thank Omar Gil, who has been a great and patient teacher, and Gregory Randall for his constant support and encouragement. FM performed part of this work while visiting the University of Minnesota. This work was partially supported by a grant from the Office of Naval Research ONR-N00014-97-1-0509, the Office of Naval Research Young Investigator Award, the Presidential Early Career Awards for Scientists and Engineers (PECASE), a National Science Foundation CAREER Award, by the National Science Foundation Learning and Intelligent Systems Program (LIS), by the Comisión Sectorial de Investigación Científica (CSIC), the Comisión Académica de Posgrado (CAP), and Tecnocom through a scholarship to FM.

A Distance Maps in Euclidean Space

We now present a few important results on distance maps. These has been mainly adapted (and adopted) from [4, 5, 25, 53].

A.1 General Properties

Wherever ψ is smooth we know that it satisfies the *Eikonal* equation

$$\|\nabla\psi\| = 1 \tag{23}$$

The distance function satisfies this PDE everywhere in the *viscosity sense* [28, 19]. It is also well known that within a sufficiently small neighborhood of $\mathcal{S} = \{\psi = 0\}$, $\psi(\cdot)$ is *smooth*, or at least as smooth as $\mathcal{S}$. These assertions can be made precise through the following Lemma from [23]:

Lemma 4 *Let $\mathcal{S}$ be a C^k ($k \geq 2$) codimension 1 closed hypersurface of $\mathbb{R}^d$. Then, the signed distance function to $\mathcal{S}$ is $C^k(U)$ for a certain neighborhood U of $\mathcal{S}$.*

Differentiating $\|\nabla\psi\|^2 = 1$ we obtain

$$D(\nabla\psi)\,\nabla\psi = 0$$

Therefore,

$$\mathbf{H}_\psi \, \nabla\psi = 0 \tag{24}$$

meaning that the normal to $\mathcal{S}$ at p is an eigenvector of the Hessian, associated to the null eigenvalue. Differentiating again we obtain

$$D^3\psi \, \nabla\psi \; + \; (D^2\psi)^2 = 0 \tag{25}$$

The next Lemma, whose detailed proof can be found in [4], is mainly based in the relations (24) and (25), and it is used to verify that the function $\mu : (-\varepsilon, \varepsilon) \to I\!R^{d\times d}$ defined by $\mu(t) = \mathbf{H}_\psi(p_0 + t\nabla\psi(p_0))$ (p_0 is any point in the manifold $\{\psi = 0\}$) satisfies the following **ODE**:

$$\dot{\mu}(t) + \mu^2(t) = 0 \quad t \in (-\varepsilon, \varepsilon)$$

Lemma 5 *The eigenvectors of $\mathbf{H}_\psi$ are constant along the characteristic lines $x(s) = x_0 + s\nabla\psi(x(s))$ (arc length parametrized, x_0 is a point onto $\mathcal{S}$) of ψ within any neighborhood where it is smooth, and the eigenvalues vary according to*

$$\lambda_i(s) = \frac{\lambda_i(0)}{s\,\lambda_i(0) + 1}$$

We use the above formula to bound the maximum offset $|\epsilon|$ of $\{\psi = 0\}$ that keeps $\{\psi = \epsilon\}$ smooth, we just take $|\epsilon| \, (\max_{1\le i\le d-1} |\lambda_i(0)|) < 1$.

We now obtain bounds on the eigenvalues of the Hessian of the distance function:

Corollary 2 *The eigenvalues $\lambda_i(p)$ of $\mathbf{H}_\psi(p)$ (principal curvatures of $\{x : \psi(x) = \psi(p)\}$) are absolutely bounded by*

$$|\lambda_i(p)| \le \frac{\mathcal{M}_\mathcal{S}}{1 - |\psi(p)|\mathcal{M}_\mathcal{S}}$$

where $\mathcal{M}_\mathcal{S}$ absolutely bounds all eigenvalues of $\mathbf{H}_\psi(p)$, $p \in \mathcal{S}$; and $|\psi(p)|$ is sufficiently small.

To conclude, let's present some concepts on projections onto the implicit surface $\mathcal{S}$, zero level-set of the distance function ψ. It is clear that the projection of a point $p \in I\!R^d$ onto $\mathcal{S}$ is given by

$$\Pi_\mathcal{S}(p) = p - \psi(p)\nabla\psi(p).$$

This projection is well defined as long as there is only one $x \in \mathcal{S}$ such that $\Pi_\mathcal{S}(p) = x$. This can be guaranteed when working within a small tubular neighborhood of a smooth surface $\mathcal{S}$. Moreover, this map is smooth within a certain tubular neighborhood of $\mathcal{S}$ [53]:

Theorem 3 *If $\mathcal{S} \subset I\!R^d$ is a compact C^k ($k \ge 2$) codimension 1 hypersurface, then there is a $h(\mathcal{S}) > 0$ such that the map $\Pi_\mathcal{S}$ is well defined and belongs to $C^{k-1}(\{x : d(x, \mathcal{S}) < h\}, I\!R^d)$.*

B Technical Lemma

Lemma

Let $f : [a,b] \to I\!R$ be a $C^1([a,b])$ function such that f' is Lipschitz. Let $\varphi \in L^\infty([a,b])$ denote (one of) f''s weak derivative. Then one has:

$$\int_a^b f'^2(x)\,dx = f\,f'|_a^b \; - \int_a^b f(x)\varphi(x)\,dx$$

Proof:
Let $ext(f')$ denote the Lipschitz extension of f' to all $I\!R$ given by

$$ext(f')(x) = \begin{cases} f'(a) \; for \; x < a \\ f'(x) \; for \; x \in [a,b] \\ f'(b) \; for \; x > b \end{cases}$$

Then let $ext(f)$ be given by any (bounded) primitive of $ext(f')$, that is $ext(f) = \int ext(f')$. Let $\widehat{\varphi} \in L^\infty I\!R$ denote $ext(f')$'s weak derivative, and we have that $\widehat{\varphi}|_{[a,b]}$ and φ coincide as weak derivatives of f. Let $\{\eta_\epsilon(\cdot)\}_{\{\epsilon>0\}}$ be a family of bounded support mollifiers. Then we define the function

$$f_\epsilon = ext(f) * \eta_\epsilon$$

It is clear that we will have

(a)

$$f_\epsilon \overset{\epsilon\downarrow 0}{\rightrightarrows} ext(f) \; over \; compact \; sets \; of \; I\!R$$

(b)

$$f'_\epsilon \overset{\epsilon\downarrow 0}{\rightrightarrows} ext(f') \; over \; compact \; sets \; of \; I\!R$$

(c)

$$f''_\epsilon \overset{\epsilon\downarrow 0}{\longrightarrow} \widehat{\varphi} \; locally \; in \; L^2(I\!R)$$

Since $f'_\epsilon \in C^\infty(I\!R)$ we can use integration by parts to conclude that:

$$\int_a^b f'^2_\epsilon(x)\,dx \; = \; f'_\epsilon f_\epsilon|_a^b \; - \int_a^b f_\epsilon(x)\,f''_\epsilon(x)\,dx$$

Now the left hand side will converge to $\int_a^b f'^2(x)\,dx$ in view of (b); the first term in the right hand side will obviously converge to $f\,f'|_a^b$. For the remaining term we observe the following ,using Cauchy-Schwartz inequality (let $<,>: L^2([a,b]) \times L^2([a,b]) \to I\!R$ denote $L^2([a,b])$'s internal product):

$$\begin{aligned} & \left| \int_a^b f_\epsilon(x)\,f''_\epsilon(x)\,dx \; - \; \int_a^b f(x)\varphi(x)\,dx \right| \\ = \; & \left| < f''_\epsilon, f_\epsilon > \; - \; < \varphi, f > \right| = \left| < f''_\epsilon, f_\epsilon - f > \; + \; < f, f''_\epsilon - \varphi > \right| \\ \leq \; & (b-a) \left(\left(\max_{\{x\in[a,b]\}} \left| f''_\epsilon(x) \right| \right) \; \| f_\epsilon - f \|_{L^2([a,b])} \; + \; \left(\max_{\{x\in[a,b]\}} |f(x)| \right) \; \| f''_\epsilon - \varphi \|_{L^2([a,b])} \right) \end{aligned}$$

Now, everything is under control since

$$\max_{\{x\in[a,b]\}} \left| f''_\epsilon(x) \right| \leq \|\varphi\|_{L^\infty([a,b])}$$

Hence we have proved

$$\int_a^b f_\epsilon(x)\, f''_\epsilon(x)\, dx \;\xrightarrow{\epsilon\downarrow 0}\; \int_a^b f(x)\varphi(x)\, dx$$

the last step of the proof.

■

References

[1] D. Adalsteinsson and J.A. Sethian, "A fast level set method for propagating interfaces," *J. Comp. Physics* **118**, pp. 269-277, 1995.

[2] R. Alexander & S. Alexander, "Geodesics in Riemannian manifolds with boundary," *Indiana University Mathematics Journal* **30:4**, 1981.

[3] S. Alexander, I.D. Berg, and R.L. Bishop, "The Riemannian obstacle problem," *Illinois Journal of Mathematics* **31:1**, Spring 1987.

[4] L. Ambrosio and N. Dancer, *Calculus of Variations and Partial Differential Equations*, Topics on Geometrical Evolution Problems and Degree Theory, Springer-Verlag, New York, 2000.

[5] L. Ambrosio and H.M. Soner, "Level set approach to mean curvature flow in arbitrary codimension," *J. Diff. Geom.* **43**, pp. 693-737, 1996.

[6] A. Bartesaghi and G. Sapiro, "A system for the generation of curves on 3D brain images," *Human Brain Mapping*, to appear.

[7] T. J. Barth and J. A. Sethian, "Numerical schemes for the Hamilton-Jacobi and level set equations on triangulated domains," *Journal of Computational Physics* **145**, pp. 1-40, 1998.

[8] M. Bertalmio, L. T. Cheng, S. Osher, and G. Sapiro, "Variational problems and partial differential equations on implicit surfaces: The framework and examples in image processing and pattern formation," *IMA University of Minnesota Preprint*, June 2000.

[9] Blitz++ website: `www.oonumerics.org/blitz`

[10] J. Bloomenthal, *Introduction to Implicit Surfaces*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997.

[11] M. Boue and P. Dupuis, "Markov chain approximation for deterministic control problems with affine dynamics and quadratic cost in the control," *SIAM J. Numer. Anal.* **36**, pp. 667-695, 1999.

[12] A. M. Bruckstein, "On shape from shading," *Comp. Vision Graph. Image Processing* **44**, pp. 139-154, 1988.

[13] V. Caselles, R. Kimmel, G. Sapiro, and C. Sbert, "Minimal surfaces based object segmentation," *IEEE-PAMI*, **19:4**, pp. 394-398, April 1997.

[14] S. Chen, B. Merriman, S. Osher, and P. Smereka, "A simple level set method for solving Stefan problems," *Journal of Computational Physics* **135**, pp. 8, 1995.

[15] L. T. Cheng, *The Level Set Method Applied to Geometrically Based Motion, Material Science, and Image Processing, PhD Thesis Dissertation, UCLA*, June 2000.

[16] L. T. Cheng, P. Burchard, B. Merriman, and S. Osher, "Motion of curves constrained on surfaces using a level set approach," *UCLA CAM Report* **00-32**, September 2000.

[17] L. D. Cohen, and R. Kimmel, "Global minimum for active contours models: A minimal path approach," *International Journal of Computer Vision* **24**, pp. 57-78, 1997.

[18] M.G. Crandall and P.L. Lions "Two approximations of solutions of Hamilton-Jacobi equations," *Math. of Comp.* **43:167**, pp. 1-19, July 1984.

[19] M.G. Crandall and P.L. Lions, "Viscosity solutions of Hamilton-Jacobi equations," *Transactions of the American Mathematical Society* **277:1**, May 1983.

[20] E. Dijkstra, "A note on two problems in connection with graphs," *Numerische Math.* **1**, pp. 269-271, 1959.

[21] M. Desbrun, M. Meyer, P. Scröeder, and A. Barr, "Discrete differential-geometry operators in nD," *Caltech, USC Report*, July 22, 2000.

[22] S. Fomel, "A variational formulation of the fast marching Eikonal solver," *Stanford Exploration Project*, Report SERGEY, May 1, pp. 357-376, May 2000.

[23] R.L. Foote, "Regularity of the distance function," *Proceedings of American Mathematical Society* **92:1**, September 1984.

[24] S. F. Frisken, R. N. Perry, A. Rockwood, and T. Jones, "Adaptively sampled fields: A general representation of shape for computer graphics," *Computer Graphics (SIGGRAPH)*, New Orleans, July 2000.

[25] J. Gomes and O. Faugeras, "Representing and evolving smooth manifolds of arbitrary codimension embedded in $\mathbb{R}^n$ as the intersection of n hypersurfaces: The vector distance functions," *RR-4012* INRIA, Oct. 2000.

[26] J. Helmsen, E. G. Puckett, P. Collela, and M. Dorr, "Two new methods for simulating photolithography development in 3D," *Proc. SPIE Microlithography* **IX**, pp. 253, 1996.

[27] P. Hoch and M. Rascle "Hamilton-Jacobi equations on a manifold and applications to grid generation or refinement," `http://www-math.unice.fr/~hoch/psfiles/hamja.ps`

[28] H. Ishii, "A simple direct proof of uniqueness for solutions of the Hamilton-Jacobi equations of Eikonal type," *Proc. Amer. Math. Soc.* **100:2**, pp. 247-151, 1987.

[29] N. Khaneja, M.I. Miller, and U. Grenander, "Dynamic programming generation of geodesics and sulci on brain surfaces," *IEEE Trans. on Pattern Analysis and Machine Intelligence* **20:11**, pp.1260-1265, November, 1998.

[30] R. Kimmel, "Numerical geometry of images: Theory, algorithms, and applications," *Technion CIS Report* **9910**, October 1999.

[31] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," *Proc. National Academy of Sciences* **95:15**, pp. 8431-8435, 1998.

[32] N. Kiryati and G. Székely, "Estimating shortest paths and minimal distances on digitized three dimensional surfaces," *Pattern Recognition* **26**, pp. 1623–1637, 1993.

[33] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," *Computer Graphics*, pp. 313-324, 1996.

[34] J. C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[35] F. Lafon and S. Osher, "High order two dimensional nonoscillatory methods for solving Hamilton-Jacobi scalar equations," *Journal of Computational Physics* **123**, pp. 235-253, 1996.

[36] J. M. Lee, *Riemannian Manifolds, An Introduction to Curvature,* Springer-Verlag, New York, Inc., 1997.

[37] C. Mantegazza and A.C. Menucci, "Hamilton-Jacobi equations and distance functions on Riemannian manifolds," `http://cvgmt.sns.it/papers/manmen99/`.

[38] A. Marino and D. Scolozzi, "Geodetiche con ostacolo," *Boll. Un. Mat. Ital.* **6:2-B**, pp. 1-31, 1983.

[39] S. Mauch, "Closest point transform,"
`www.ama.caltech.edu/~ seanm/software/cpt/cpt.html`.

[40] F. Memoli, G. Sapiro, and S. Osher, "Harmonic maps onto implicit manifolds," pre-print, March 2001.

[41] J. S. B. Mitchell, "An algorithmic approach to some problems in terrain navigation," *Artificial Intelligence* **37**, pp. 171-201, 1988.

[42] J. S. B. Mitchell, D. Payton, and D. Keirsey, "Planning and reasoning for autonomous vehicle control," *International Journal of Intelligent Systems* **2**, pp. 129-198, 1987.

[43] S. Osher, "A level-set formulation for the solution of the Dirichlet problem for Hamilton-Jacobi equations," *SIMA J. Numer. Anal.* **24**, pp. 1145, 1993.

[44] S. J. Osher and R. P. Fedkiw, "Level set methods," *ULCA CAM Report* **00-07**, February 2000.

[45] S. J. Osher and J. A. Sethian, "Fronts propagation with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics* **79**, pp. 12-49, 1988.

[46] D. Peng, B. Merriman, S. Osher, H. Zhao, M. Kang, "A PDE-based fast local level set method," *Journal of Computational Physics* **155**, pp. 410-438, 1999.

[47] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Texts and Monographs in Computer Science, Springer-Verlag, New York, 1990.

[48] E. Rouy and A. Tourin, "A viscosity solutions approach to shape-from-shading," *SIAM. J. Numer. Analy.* **29:3**, pp. 867-884, 1992.

[49] T. Sakai, *Riemannian Geometry*, AMS Translations of Mathematical Monographs **149**.

[50] J. Sethian, "Fast marching level set methods for three-dimensional photolithography development," *Proc. SPIE International Symposium on Microlithography*, Santa Clara, California, March, 1996.

[51] J. A. Sethian, "A fast marching level-set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci.* **93:4**, pp. 1591-1595, 1996.

[52] J. A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Sciences,* Cambridge University Press, Cambridge-UK, 1996.

[53] L. Simon, *Theorems on Regularity and Singularity of Energy Minimizing Maps*, Birkhäuser, Boston, 1996.

[54] G. Taubin, "Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation," *IEEE Trans. PAMI* **13:11**, pp. 1115-1138, 1991.

[55] A. W. Toga, *Brain Warping*, Academic Press, New York, 1998.

[56] P. Thompson, R. Woods, M. Mega, and A. Toga, "Mathematical/computational challenges in creating deformable and probabilistic atlases of the human brain," *Human Brain Mapping* **9:2**, pp. 81-92, 2000.

[57] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control* **40** pp. 1528-1538, 1995.

[58] D. C. Van Essen, H. Drury, S. Joshi and M. I. Miller, "Functional and structural mapping of human cerebral cortex: Solutions are in the surfaces," *Proceedings of the National Academy of Science* **95**, pp. 788-795, February 1998.

[59] B. Wandell, S. Chial, and B. Backus, "Cortical visualization," *Journal of Cognitive Neuroscience*, to appear.

[60] A. Witkin and P. Heckbert, "Using particles to sample and control implicit surfaces," *Computer Graphics (SIGGRAPH)*, pp. 269-278, 1994.

[61] F.E. Wolter, "Cut loci in bordered and unbordered Riemannian manifolds," *Doctoral Dissertation, Technische Universität Berlin*, 1985.

[62] Z. Wood, M. Desburn, P. Schröder, and D. Breen, "Semi-Regular mesh extraction from volume," *Proc. IEEE Visualization*, 2000.

[63] G. Yngve and G. Turk, "Creating smooth implicit surfaces from polygonal meshes," *Technical Report GIT-GVU-99-42, Graphics, Visualization, and Usability Center. Georgia Institute of Technology*, 1999 (obtained from `www.cc.gatech.edu/gvu/geometry/publications.html`).

[64] A. Yezzi, S. Kichenassamy, P. Olver, and A. Tannenbaum, "Geometric active contours for segmentation of medical imagery," *IEEE Trans. Medical Imaging* **16**, pp. 199-210, 1997.

[65] H. Zhao, S. Osher, B. Merriman, and M. Kang, "Implicit, non-parametric shape reconstruction from unorganized points using a variational level set method," *Comp. Vision and Image Understanding* **80**, pp. 295-314, 2000.

[66] G. Zigelman, R. Kimmel, and N. Kiryati, "Texture mapping using surface flattening via multi-dimensional scaling," *Technion-CIS Technical Report* **2000-01**, 2000.

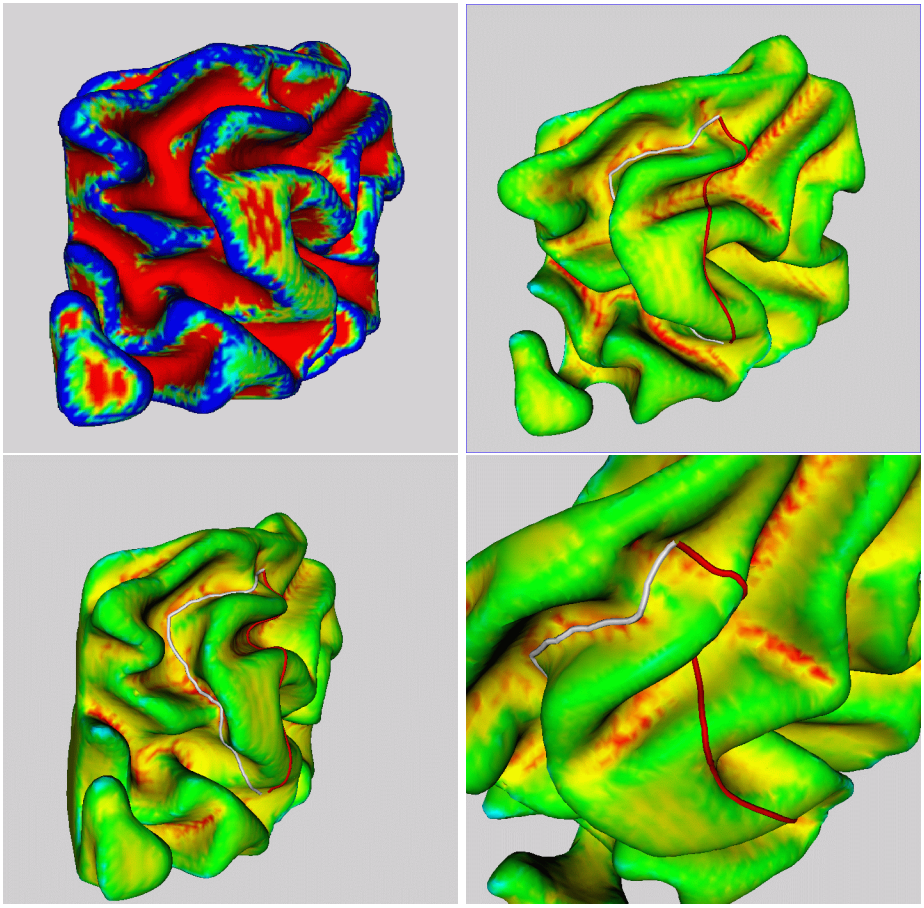


Figure 10: These four figures show the detection of valleys over implicit surfaces representing a portion of the human cortex. We use a mean curvature based weighted distance. In the left-upper corner we show the mean curvature of the brain surface (clipped to improve visualization). It is quite convincing that this quantity can be of of great help to detect valleys. In the remaining figures we show two curves over the surface, whose coloring correspond to the mean curvature (not clipped, from red, yellow, green to blue, as the value increases). The red curve is the one that corresponds to the *natural* geodesic ($g = 1$), while the white curve is the weighted-geodesic that should travel through "nether" regions. Indeed, a very clear difference exists between both trajectories, since the white curve makes its way through regions where the mean curvature attains low values. The figure in the right-low quadrant is a zoomed view of the same situation.

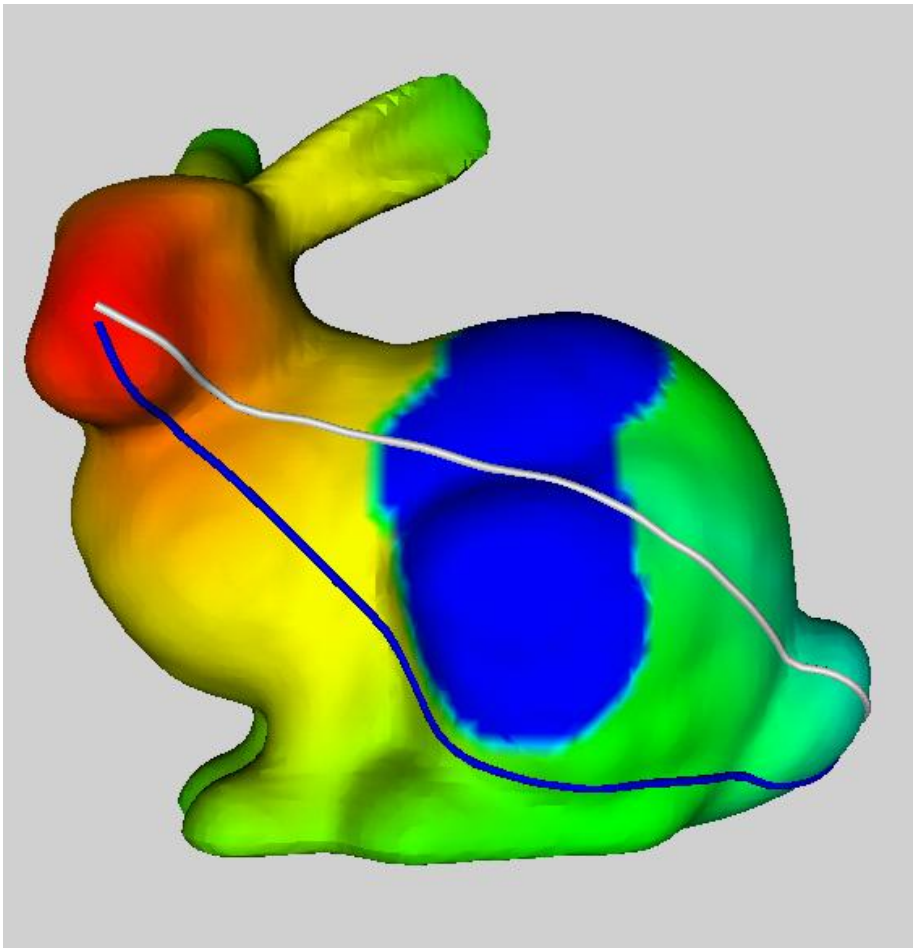


Figure 11: Distance map and geodesic curve between two points on an implicit bunny surface with an intrinsic obstacle on it. We now use a binary weight, $g = \{1, \infty\}$, being infinity at the obstacle. This permits, as illustrated in the figure, the computation of optimal paths with obstacles on implicit surfaces. The blue path corresponds to the obstacle-weighted distance function, and the white one to the natural ($g = 1$) distance function. Both geodesics are shown over the surface of the bunny, the pseudocolor representing the weighted distance for the surface with obstacle. The obstacle is also shown in blue.

Solving PDE's on Implicit Surfaces Good Bye Triangulated Surfaces?

Guillermo Sapiro

Electrical and Computer Engineering

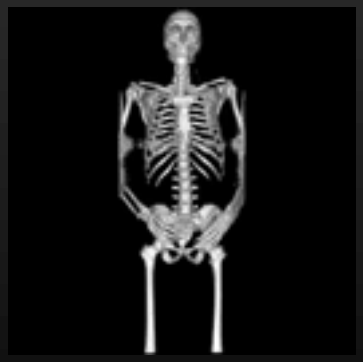
University of Minnesota

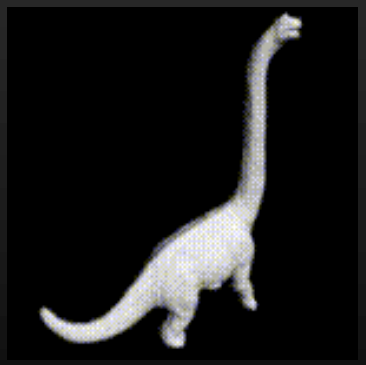
guille@ece.umn.edu

Supported by NSF, ONR, PECASE, CAREER, NIH

The problems

- Solve PDE's and variational problems for data defined from a generic surface onto a generic surface



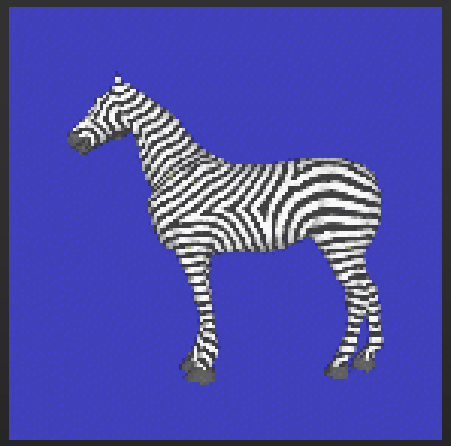


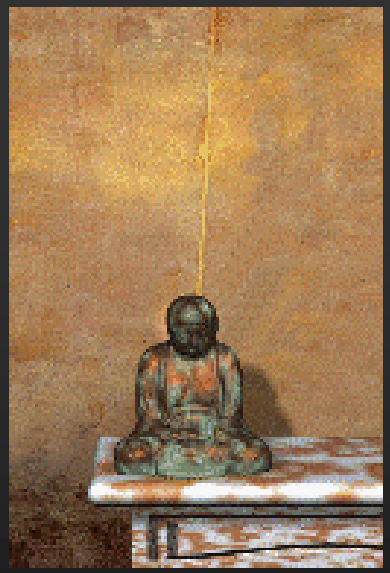

- Surfaces from the data bases at Stanford and GATECH

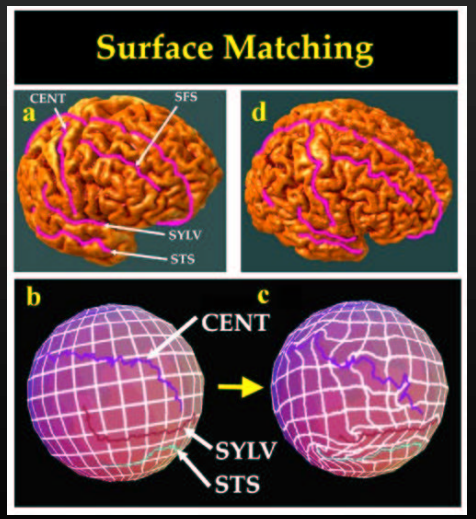
Variational problems and PDE's on surfaces?

- Mean curvature motion (Ilmanen, etc)
- Mathematical physics
- Computer graphics
- Image processing
- Regularization of inverse problems (e.g., EEG+MRI)
- 3D surface mapping

Examples



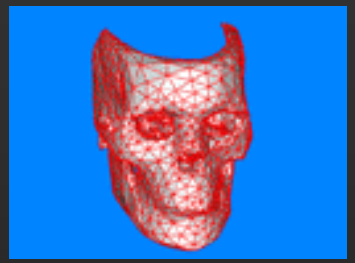




- Images from G. Turk , J. Dorsey, P. Thompson

Classical approach

- **Work with triangulated/meshed surfaces**
  - Discretization on non-uniform grids
  - Projections onto triangulated surfaces



- **Limit to functions (e.g., Kimmel)**
  - Limited framework
- **Work on surfaces mapped to the plane**
  - Loose the geometry, ads complexity
- **No work on target surfaces reported**

Our approach

- **Define the intrinsic variational problems and PDE's (e.g., following the theory of Harmonic Maps)**
- **Represent the surfaces in implicit form**
  - Classical numerics on Cartesian grids
  - No projections
  - Motivated by Osher-Sethian level-sets and Osher variational levels-sets (though here the surface is not moving)

The embedding

- **I:M -> N**
  - M is a generic surface (domain)
    - With Bertalmio, Cheng, Osher
  - N is a generic surface (target)
    - With Memoli, Osher

Surface in implicit form

- Original data
- Via polynomials (Taubin)
- From range data (Zhao et al., Levoy, ...)
- From triangulated surfaces (Eck, Yngve,...)

Implicit form

$$M = \text{level} - \text{set of } \Psi = \{x : \Psi(x) = 0\}$$

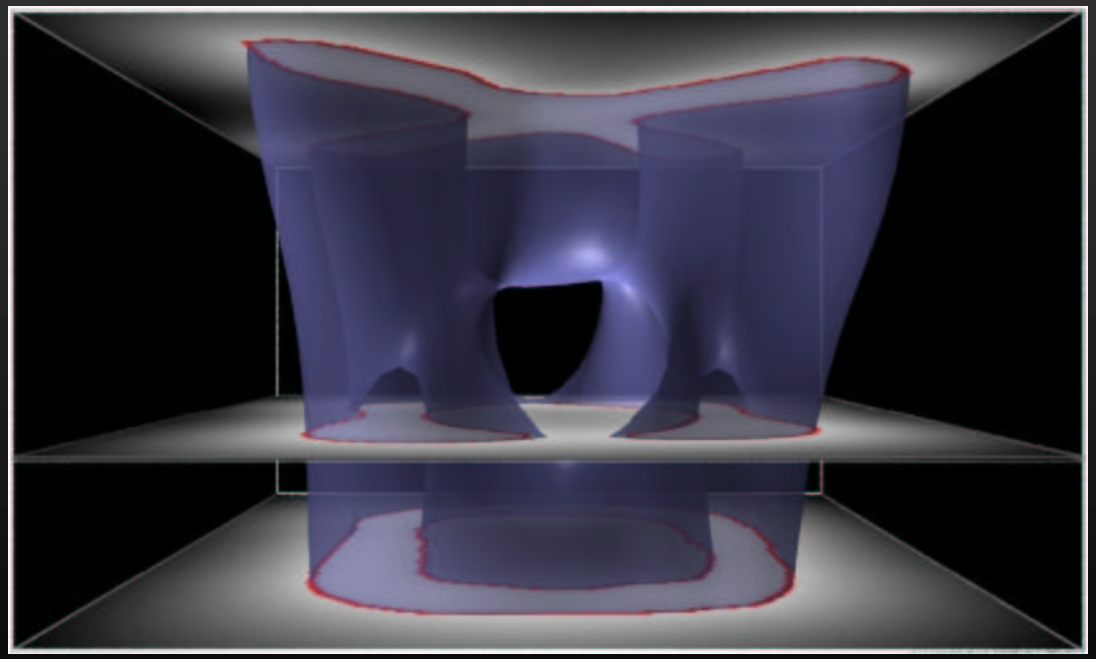


Figure from G. Turk

Data extension

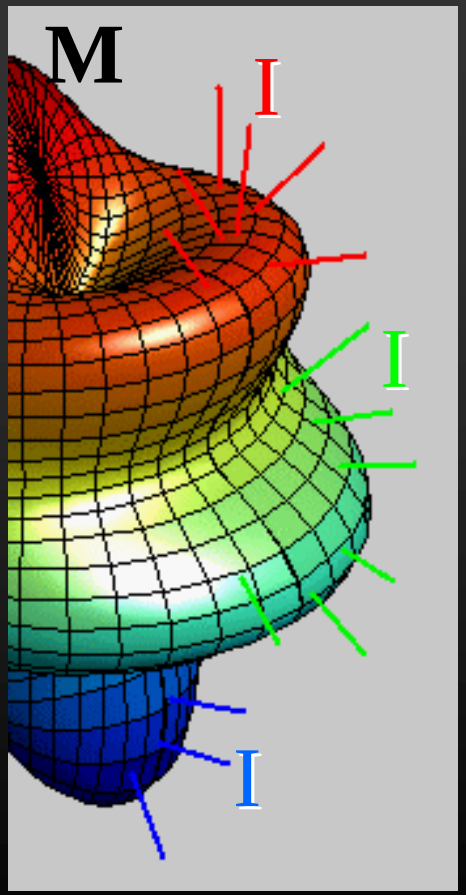
$$\mathrm{I} : \mathrm{M} \rightarrow \mathrm{R}$$

- **Embed M:**

$$\mathrm{M} = \{x : \Psi(x) = 0\}$$

- **Extend I outside M:**

$$\frac{\partial \mathrm{I}}{\partial \mathrm{t}} + sign(\psi)\,(\nabla I \cdot \nabla \psi) = 0$$



Heat flow on the plane

$$E = \int \|\nabla I\|^2 \Rightarrow \frac{\partial I}{\partial t} = \Delta I$$

Embedding the domain surface

- **Example: I:M->R**
  - A map from a generic domain surface onto the real line

$$\min_{I:M\to R} \int_M \left\| \nabla_M I \right\|^2 \, dvol_M$$

$$\frac{\partial I}{\partial t} = \Delta_M I$$

Embedding the domain surface (cont.)

$$\int_M \left\| \nabla_M I \right\|^2 dvol_M = \int_M \left\| P_{\nabla\Psi} \nabla I \right\|^2 dvol_M$$

$$= \int_{R^3} \left\| P_{\nabla\Psi} \nabla I \right\|^2 \delta(\Psi) \left\| \nabla\Psi \right\| d\mathbf{x}$$

Embedding the domain surface (cont.)

- The gradient descent flow: Heat flow on intrinsic surfaces

$$\frac{\partial I}{\partial t} = \frac{1}{\|\nabla \psi\|} \operatorname{div}\left(P_{\nabla \Psi} \nabla I \, \|\nabla \psi\|\right)$$

- All the computations are done in the Cartesian grid!

Framework

- If embedding with distance function:

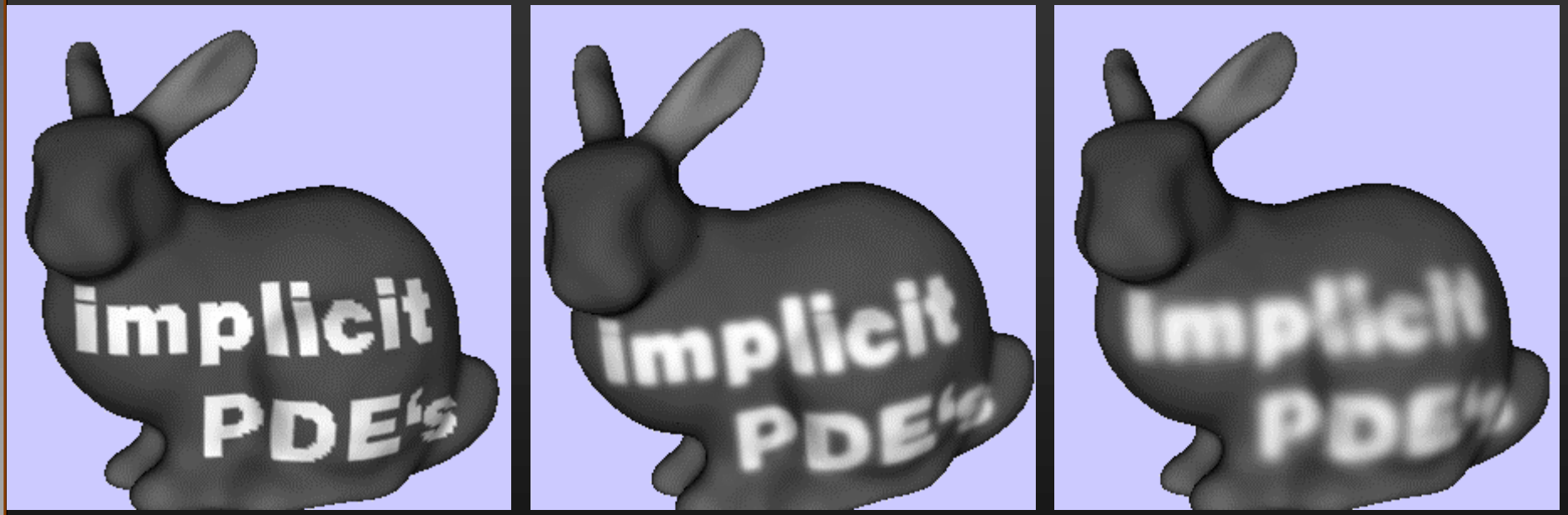
$$\frac{\partial I}{\partial t} = \operatorname{div}\left(P_{\nabla\Psi} \nabla I\right)$$

- Compare with planar case:

$$\frac{\partial I}{\partial t} = \operatorname{div}\left(\nabla I\right) = \Delta I$$

Example

Example: intrinsic heat flow



L1 denoising on implicit surfaces

$$\int_M \|\nabla_M I\| \, ds = \int_M \|P_{\nabla\Psi} \nabla I\| \, ds$$

$$= \int_{R^3} \|P_{\nabla\Psi} \nabla I\| \delta(\bar{\Psi}) \|\nabla\Psi\| \, dx$$

$$\frac{\partial I}{\partial t} = \frac{1}{\|\nabla\Psi\|} \operatorname{div}\left(\frac{P_{\nabla\Psi} \nabla I}{\|P_{\nabla\Psi} \nabla I\|} \|\nabla\Psi\| \right)$$

L1 denoising on implicit surfaces

$$\frac{\partial \mathrm{I}}{\partial \mathrm{t}} = \frac{1}{\|\nabla\Psi\|}\,\mathrm{div}\left(\frac{P_{\nabla\Psi}\nabla I}{\|P_{\nabla\Psi}\nabla I\|}\|\nabla\Psi\|\right)$$

intrinsic

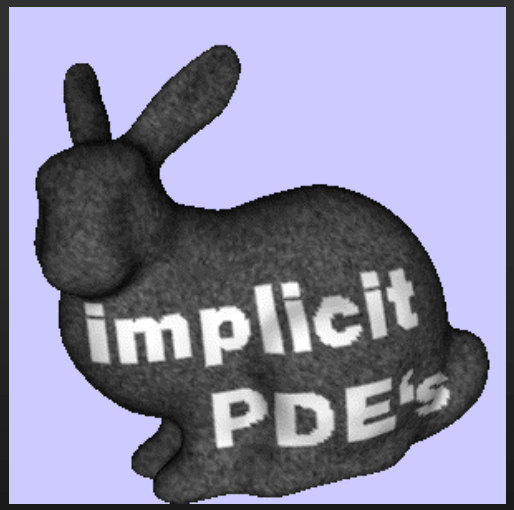
$$\frac{\partial \mathrm{I}}{\partial \mathrm{t}} = \mathrm{div}\left(\frac{P_{\nabla\Psi}\nabla I}{\|P_{\nabla\Psi}\nabla I\|}\right)$$

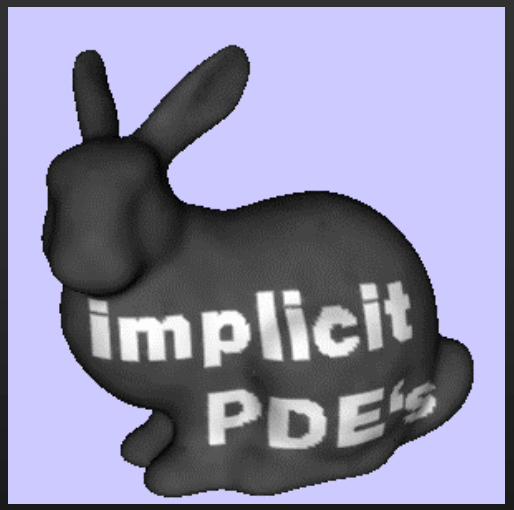
intrinsic, embedding w/ signed distance

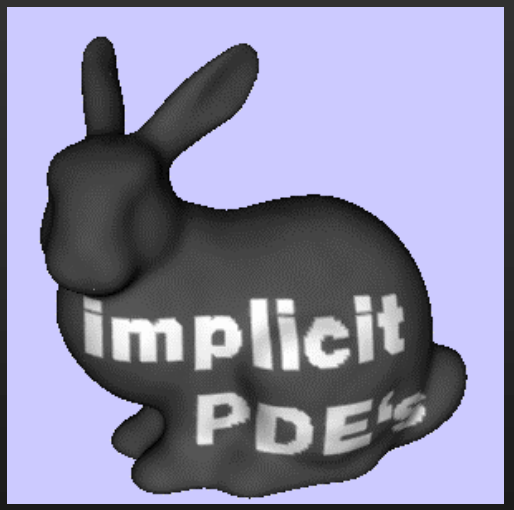
$$\frac{\partial \mathrm{I}}{\partial \mathrm{t}} = \mathrm{div}\left(\frac{\nabla I}{\|\nabla I\|}\right)$$

planar

Example:
L1 denoising with constraints







Unit vector/color denoising on implicit surfaces

- I is a map from the 3D surface to the 3D unit sphere

$$\frac{\partial \mathbf{I}}{\partial t} = \frac{1}{\|\nabla \Psi\|} \operatorname{div}\left(\frac{P_{\nabla \Psi} \nabla I}{\|P_{\nabla \Psi} \nabla I\|} \|\nabla \Psi\| \right) + \mathbf{I} \|P_{\nabla \Psi} \nabla I\|$$

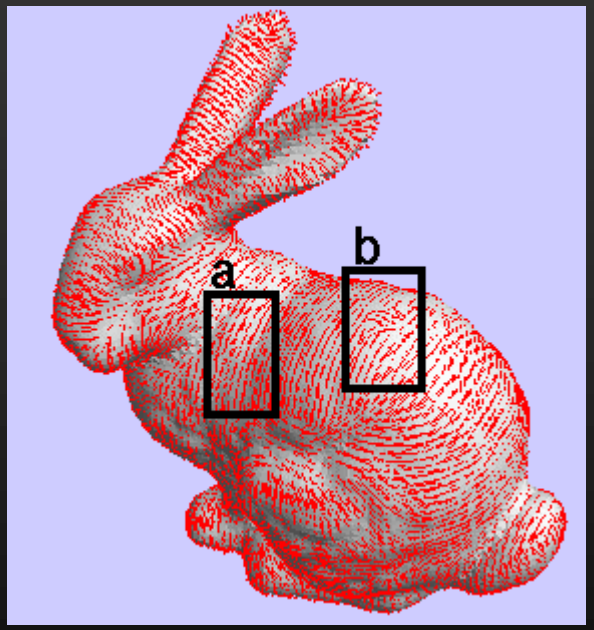
Example: Chroma denoising on a surface

original

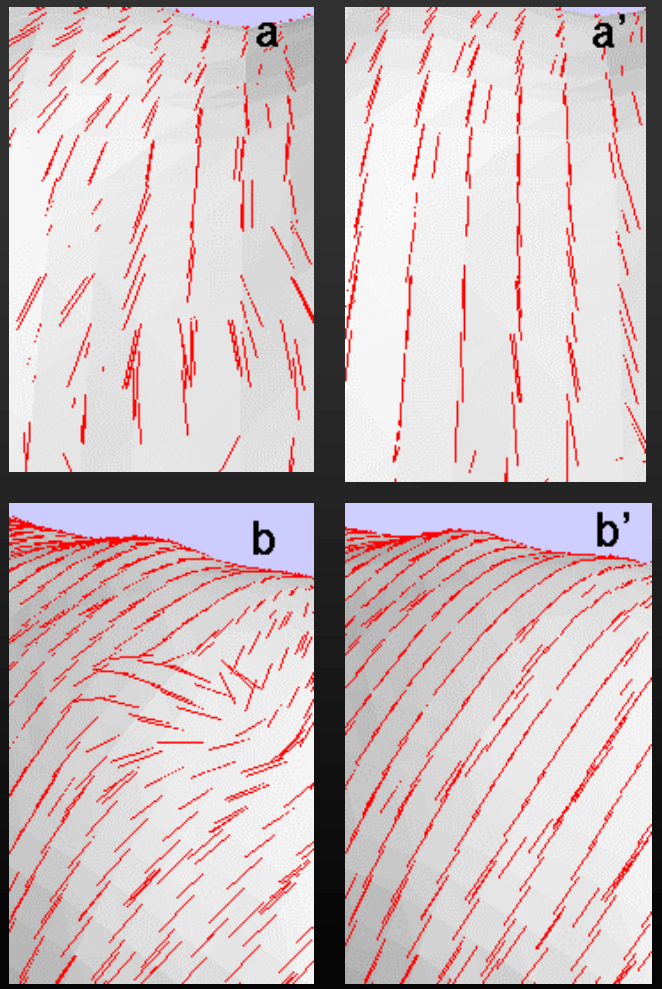
noisy

enhanced

Example: Direction denoising

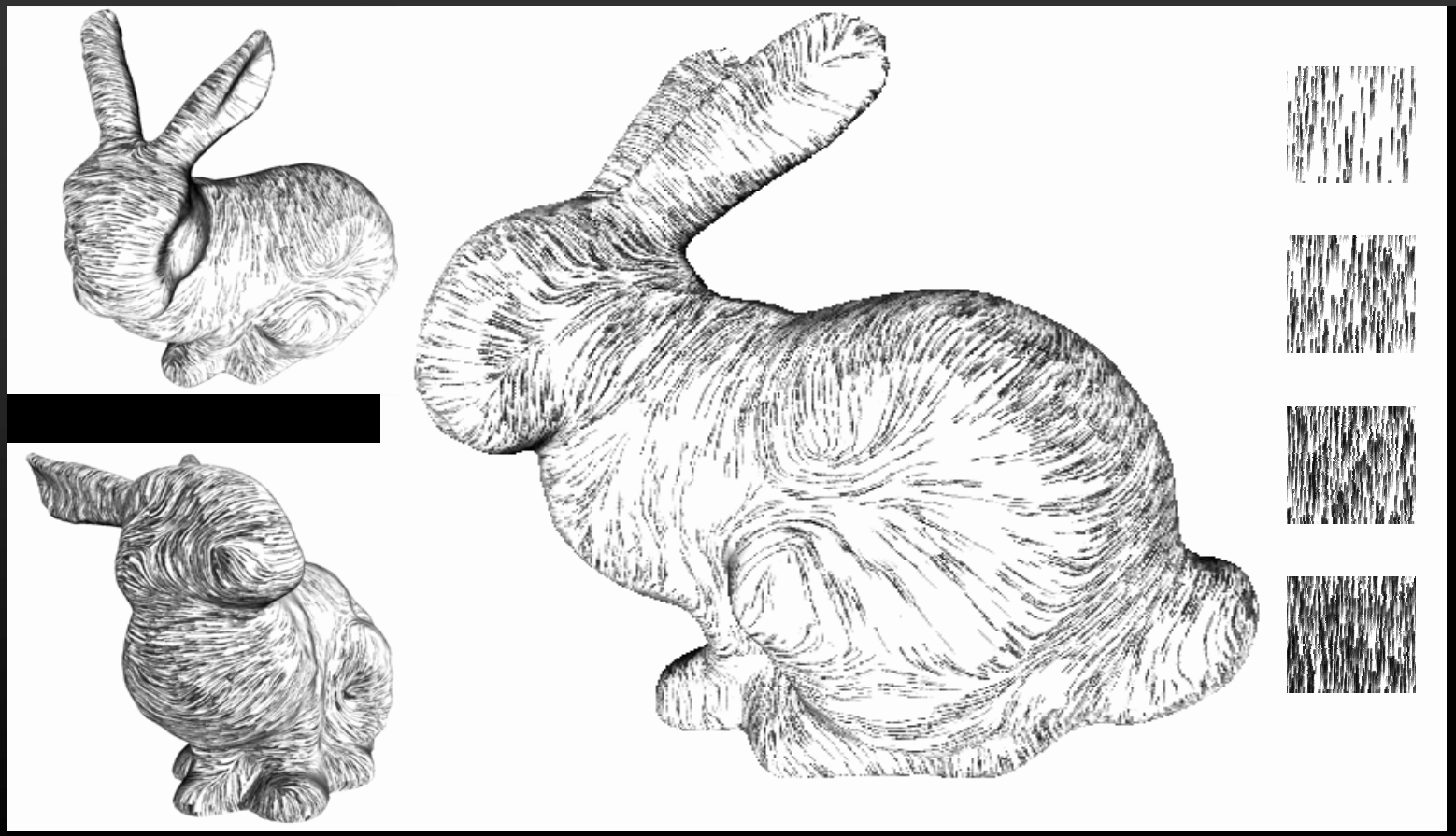


noisy



cleaned

Application (with G. Gorla and V. Interrante)

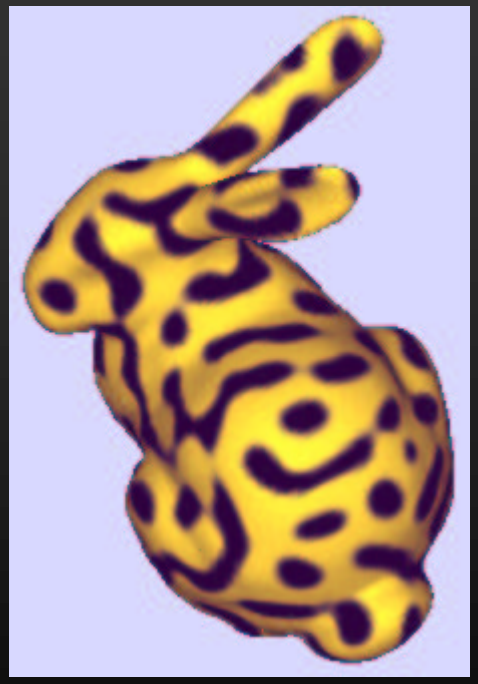


Pattern formation on implicit 3D surfaces

- Follows Turing, Kass-Witkin, Turk

$$\left.\begin{aligned}\frac{\partial a}{\partial t} &= f(a,b) + \alpha\,\Delta_M a \\ \frac{\partial b}{\partial t} &= g(a,b) + \beta\,\Delta_M b\end{aligned}\right\} \Rightarrow \begin{cases}\dfrac{\partial a}{\partial t} = f(a,b) + \alpha\,\dfrac{1}{\|\nabla\Psi\|}\,\mathrm{div}\left(\mathsf{P}_{\nabla\Psi}\mathsf{I}\,\|\nabla\Psi\|\right) \\ \dfrac{\partial b}{\partial t} = g(a,b) + \beta\,\dfrac{1}{\|\nabla\Psi\|}\,\mathrm{div}\left(\mathsf{P}_{\nabla\Psi}\mathsf{I}\,\|\nabla\Psi\|\right)\end{cases}$$

Examples

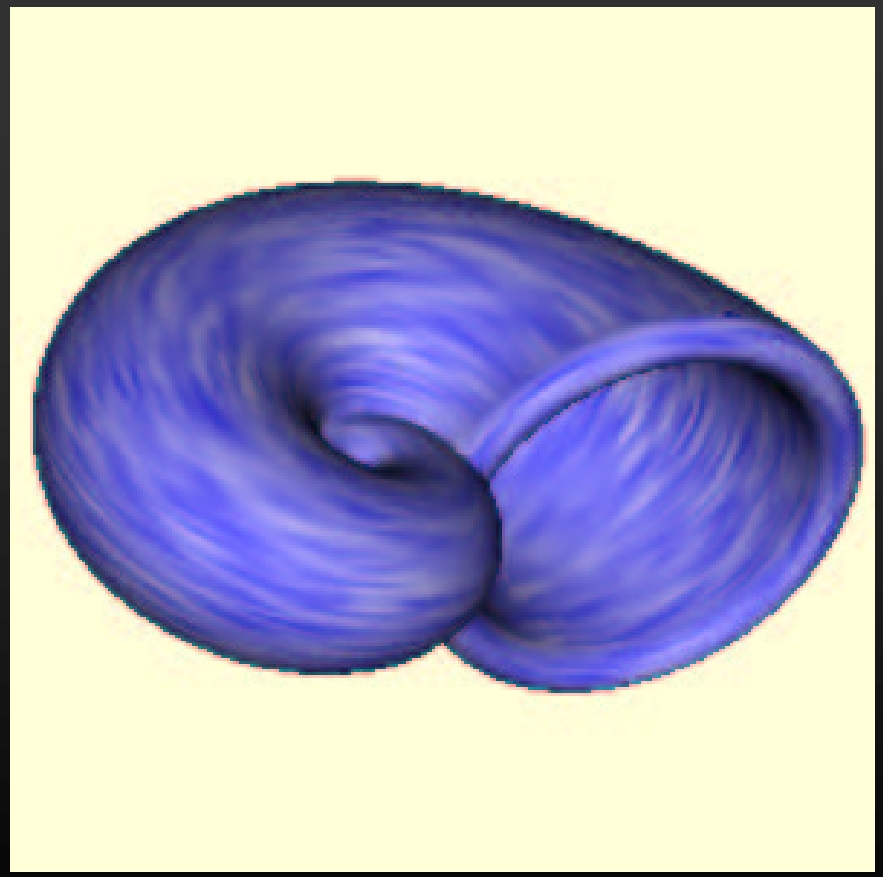


Vector field visualization

- I is random noise, diffused in direction *v*

$$\left.\begin{aligned}\frac{\partial I}{\partial t} &= \operatorname{div}(\mathrm{A}\nabla I)\\ \mathrm{A} &= \vec{v}^{T}\vec{v}\end{aligned}\right\} \Rightarrow \begin{cases}\dfrac{\partial I}{\partial t} = \dfrac{1}{\|\nabla\Psi\|}\operatorname{div}\left(\mathrm{A}\mathrm{P}_{\nabla\Psi}\nabla\mathrm{I}\,\|\nabla\Psi\|\right)\\ \mathrm{A} = \vec{v}^{T}\vec{v}\end{cases}$$

Vector field visualization (e.g., principal directions)



Embedding the target manifold

- I:M->N

$$N = \text{level} - \text{set of } \Phi = \{x : \Phi(x) = 0\}$$

$$\min_{I:M \to \{\Phi=0\}} \int_M \left\| \nabla_M I \right\|^p \, dvol_M$$

$$\frac{\partial I}{\partial t} = \Delta_M I + A_{\{\Phi=0\}}(I) < \nabla_M I, \nabla_M I >$$

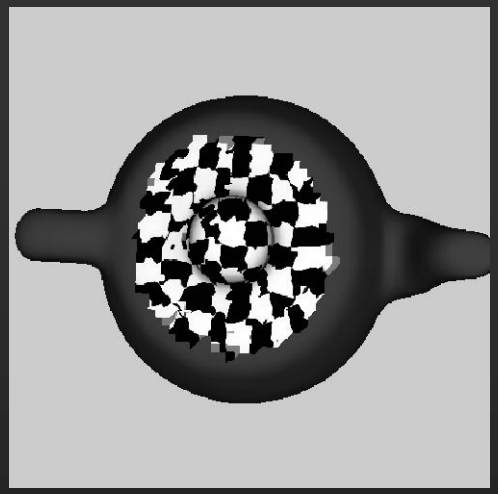
Embedding the target manifold (cont.)

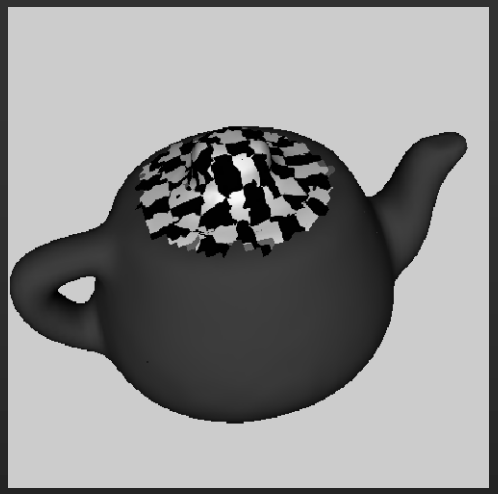
$$\min_{\mathbf{I}:R^k \to \{\Phi=0\}} \int_{R^k} \left\| \nabla_M \mathbf{I} \right\|^2 d\mathbf{x}$$

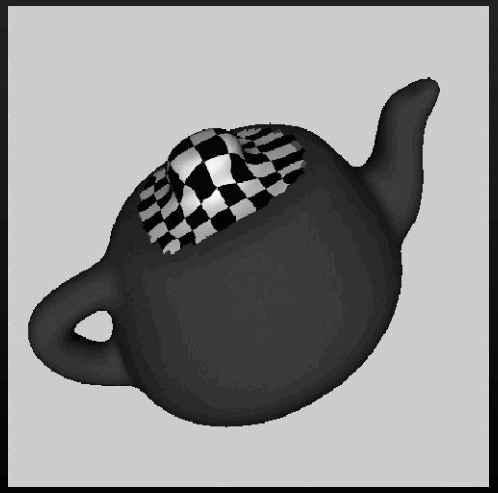
$$\frac{\partial \mathbf{I}}{\partial t} = \Delta \mathbf{I} + \left(\sum_k \mathbf{H}_\Phi \left\langle \frac{\partial \mathbf{I}}{\partial \mathbf{x}_k}, \frac{\partial \mathbf{I}}{\partial \mathbf{x}_k} \right\rangle \right) \left\| \nabla \Phi \right\|$$

Texture mapping denoising

Texture mapping denoising







Conclusions

- **Solve PDE's for data defined on implicit surfaces**
  - Cartesian grids
  - Classical numerics
- **Extended to work with general target implicit domains (Memoli-Sapiro-Osher)**

Thanks

F. Memoli and G. Sapiro, “Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces,” *Journal of Computational Physics* 173:2, pp. 730-764, November 2001.

B. Tang, G. Sapiro, and V. Caselles, “Color image enhancement via chromaticity diffusion,” *IEEE Trans. Image Processing* 10, pp. 701-707, May 2001.

B. Tang, G. Sapiro, and V. Caselles, “Diffusion of general data on non-flat manifolds via harmonic maps theory: The direction diffusion case,” *Int. Journal Computer Vision* 36:2, pp. 149-161, February 2000.

M. Bertalmio, L. T. Cheng, S. Osher, and G. Sapiro, “Variational problems and partial differential equations on implicit surfaces,’’ *Journal of Computational Physics* 174:2, pp. 759-780, 2001.

A. Bartesaghi and G. Sapiro, “A system for the generation of curves on 3D brain images,” *Human Brain Mapping* 14:1, pp. 1-15, 2001.

A Framework for Solving Surface Partial Differential Equations for Computer Graphics Applications

Marcelo Bertalmio and Guillermo Sapiro*
Electrical and Computer Engineering, University of Minnesota

Li-Tien Cheng and Stanley Osher
UCLA Mathematics Department

Abstract

A novel framework for solving variational problems and partial differential equations for scalar and vector-valued data defined on surfaces is introduced in this paper. The key idea is to implicitly represent the surface as the level set of a higher dimensional function, and solve the surface equations in a fixed Cartesian coordinate system using this new embedding function. This thereby eliminates the need for performing complicated and not-accurate computations on triangulated surfaces, as it is commonly done in the graphics and numerical analysis literature. We describe the framework and present examples in texture synthesis, flow field visualization, as well as image and vector field regularization for data defined on 3D surfaces.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—; G.1.8 [Numerical Analysis]: Partial Differential Equations—; I.4.10 [Image Processing and Computer Vision]: Image representation—;

Keywords: Partial differential equations, implicit surfaces, pattern formation, natural phenomena, flow visualization, image and vector field regularization.

1 Introduction

In a number of computer graphics applications, variational problems and partial differential equations (PDE's) need to be solved for data defined on arbitrary manifolds, three dimensional surfaces in particular. Examples of this are texture synthesis [37, 41], vector field visualization [10], and weathering [11]. In addition, data defined on surfaces often needs to be regularized, e.g., as part of a vector field computation or interpolation process [28, 38], for inverse problems [15], or for surface parameterization [12]. These last examples can be addressed by solving a variational problem on the surface, or its corresponding gradient-descent flow on the surface, using for example the theory of harmonic maps [14], which has recently been demonstrated to be of use for computer graphics applications as well, e.g., [12, 32, 43]. All these equations are generally solved on triangulated or polygonal surfaces. That is, the surface is given in polygonal (triangulated) form, and the data is discretely defined on it. This involves the non-trivial discretization of the equations in general polygonal grids, as well as the difficult numerical computation of other quantities like projections onto the discretized surface (when computing gradients and Laplacians for example). Although the use of triangulated surfaces is extremely popular in computer graphics, there is still *not* a widely accepted technique to compute differential characteristics such as tangents, normals, principal directions, and curvatures; see for example [9, 23, 34] for a few of the approaches in this direction. On the other hand, it is widely accepted that computing these objects for iso-surfaces (implicit representations) is straightforward and much more accurate and robust. This problem in triangulated surfaces becomes even bigger when we not only have to compute these first and second order differential characteristics of the surface, but also have to use them to solve variational problems and PDE's for data defined on the surface. Moreover, virtually no analysis exists on numerical PDE's on non-uniform grids in the generality needed for computer graphics, making it difficult to understand the behavior of the numerical implementation and its proximity (or lack thereof) to the continuous model.

In this paper we present a new framework to solve variational problems and PDE's for scalar and vector-valued data defined on surfaces. We use, instead of a triangulated/polygonal representation, an implicit representation: our surface will be the zero-level set of a higher dimensional *embedding* function (i.e., a 3D volume with real values, positive outside the surface and negative inside it). Implicit surfaces have been widely used in computer graphics, e.g., [3, 16, 40], as an alternative representation to triangulated surfaces. We smoothly extend the original (scalar or vector-valued) data lying on the surface to the 3D volume, adapt our PDE's accordingly, and then perform all the computations on the Cartesian grid corresponding to the embedding function. These computations are nevertheless intrinsic to the surface. The advantages of using the Cartesian grid instead of a triangulated mesh are many: we can use well studied numerical techniques, with accurate error measures; the topology of the underlying surface it is not an issue; and we can derive simple, accurate, robust and elegant implementations. If the original surface is not already in implicit form, and it is for example triangulated, we can use any of a number of implicitation algorithms that achieve this representation given a triangulated input, e.g., [13, 20, 33, 42]. For example, the public domain software [21] can be used. If the data is just defined on the surface, an extension of it to the whole volume is also easily achieved using a PDE, as we will see. Therefore, the method here proposed works as well for non-implicit surfaces after the preprocessing is performed. This preprocessing is quite simple and no complicated regridding needs to be done to go from one surface representation to another (see below). Finally, we will solve the variational problem or PDE only in a band surrounding the zero level set (a classical approach; see [26]). Therefore, although we will be increasing by one the dimension of the space, the computations remain of the same complexity, while the accuracy and simplicity are significantly improved.

1.1 Our contribution

Representing *deforming* surfaces as level sets of higher dimensional functions was introduced in [24] as a very efficient technique for numerically studying the deformation (see also [40] for studies on the deformation and manipulation of implicit surfaces for graphics applications). The idea is to represent the surface deformation via the embedding function deformation, which adds accuracy, robustness, and, as expected, topological liberty. When the velocity of the deformation is given by the minimization of an energy, the authors in [44] proposed a "variational level set" method, where

*Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA, {marcelo,guille}@ece.umn.edu

they extended the energy (originally defined only on the surface) to the whole space. This allows for the implementation to be in the Cartesian grid. The key of this approach is to go from a "surface energy" to a "volume energy" by using a Dirac's delta function that concentrates the penalization on the given surface.

We will follow this general direction with our fixed, *non deforming* surfaces. In our case, what is being "deformed" is the (scalar or vector-valued) data on the surface. If this deformation is given by an energy-minimization problem (as is the case in data smoothing applications), we will extend the definition of the energy to the whole 3D space, and its minimization will be achieved with a PDE, which despite its being intrinsic to the underlying surface, it is also defined in the whole space. Therefore, it is easily implementable. This is straightforward, as opposed to approaches where one maps the surface data onto the plane, performs the required operations there and then maps the results back onto the triangulated representation of the surface; or approaches that attempt to solve the problem directly on a polygonal surface.

Very interestingly, the new framework proposed here also tells us how to translate into surface terms PDE's that we know that work on the plane but which do not necessarily minimize an energy (e.g., texture synthesis or flow visualization PDE's). Instead of running these PDE's on the plane an then mapping the results onto a triangulated representation of the surface, or running them directly on the triangulated domain, we obtain a 3D straightforward Cartesian grid realization that implements the equation intrinsically on the surface and whose accuracy depends only on the degree of spatial resolution.

Moreover, we consider that for computing differential characteristics and solving PDE's even for triangulated surfaces, it might be appropriate to run an implicitation algorithm as any of the ones used for the examples in this paper and then work on the implicit representation. Current algorithms for doing this, some of them publically available [21], are extremely accurate and efficient.

The contribution of this paper is then a new technique to efficiently solve a common problem in many computer graphics applications: the implementation of PDE's on 3D surfaces. In particular, we show how to transform any intrinsic variational or PDE equation into its corresponding one for implicit surfaces. In this paper we are then proposing a new framework to better solve existent problems and to help in building up the solutions for new ones. To exemplify the technique and its generality, we implement and extend popular equations previously reported in the literature. Here we solve them with our framework, while in the literature were solved with elaborated discretizations on triangulated representations.

2 The general framework

As mentioned before, our approach requires us to have an implicit representation of the given fixed surface, and the data must be defined in a band surrounding it and not just on the surface. The implicit surfaces used in this paper have been derived from public-domain triangulated surfaces via the computation of a (signed) distance function $\psi(x, y, z)$ to the surface $\mathcal{S}$. Arriving at an implicit representation from a triangulated one is not an issue, there are publicly available algorithms that achieve it in a very efficient fashion. To exemplify this, in our paper we have used several of these techniques. For some surfaces the classical Hamilton-Jacobi equation $\parallel \nabla\psi \parallel = 1$ was solved on a pre-defined grid enclosing the given surface via the computationally optimal approach devised in [35]. Accurate implicit surfaces from triangulations of the order of one million triangles are obtained in less than two minutes of CPU-time with this technique. Alternatively we used the implementation of the Closest Point Transform available in [21]. The teapot and knot surfaces were obtained from unorganized data points using the technique devised in [45]. We therefore assume from now on that the three dimensional surface $\mathcal{S}$ of interest is given in implicit form, as the zero level set of a given function $\psi : I\!R^3 \rightarrow I\!R$. This function is negative inside the closed bounded region defined by $\mathcal{S}$, positive outside, Lipschitz continuous a.e., with $\mathcal{S} \equiv \{x \in I\!R^3 : \psi(x) = 0\}$. To ensure that the data, which needs not to be defined outside of the surface originally, is now defined in the whole band, one simple possibility is to extend this data u defined on $\mathcal{S}$ (i.e the zero level set of ψ) in such a form that it is constant normal to each level set of ψ. This means the extension satisfies $\nabla u \cdot \nabla\psi = 0$. (For simplicity, we assume now u to be a scalar function, although we will also address in this paper problems where the data defined on $\mathcal{S}$ is vector-valued. This is solved in an analogous fashion.) To solve this we numerically search for the steady state solution of the Cartesian PDE

$$\frac{\partial u}{\partial t} + \text{sign}(\psi)(\nabla u \cdot \nabla\psi) = 0.$$

This technique was first proposed and used in [7]. Note that this keeps the given data u on the zero level set of ψ (the given surface) unchanged.

Both the implicitation and data extension (if required at all by the given data), need to be done only once off line. Moreover, they will remain for all applications that need this type of data.

We will exemplify our framework with the simplest case, the heat flow or Laplace equation for scalar data defined on a surface. For scalar data u defined on the plane, that is, $u(x, y) : I\!R^2 \rightarrow I\!R$ it is well known that the heat flow

$$\frac{\partial u}{\partial t} = \Delta u \tag{1}$$

where $\Delta := \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ is the Laplacian, is the gradient descent flow of the Dirichlet integral

$$\frac{1}{2}\int_{I\!R^2} \parallel \nabla u \parallel^2 dxdy, \tag{2}$$

where ∇ is the gradient.

Eq. (1) performs smoothing of the scalar data u, and this smoothing process progressively decreases the energy defined in eq. (2). If we now want to smooth scalar data u defined on a surface $\mathcal{S}$, we must find the minimizer of the energy given by

$$\frac{1}{2}\int_{\mathcal{S}} \parallel \nabla_{\mathcal{S}} u \parallel^2 d\mathcal{S}, \tag{3}$$

The equation that minimizes this energy is its gradient descent flow:

$$\frac{\partial u}{\partial t} = \Delta_{\mathcal{S}} u. \tag{4}$$

Here $\nabla_{\mathcal{S}}$ is the intrinsic gradient and $\Delta_{\mathcal{S}}$ the intrinsic Laplacian or Laplace-Beltrami operator. These are classical concepts in differential geometry, and basically mean the natural extensions of the gradient and Laplacian respectively, considering all derivatives intrinsic to the surface. For instance, the intrinsic gradient is just the projection onto $\mathcal{S}$ of the regular 3D gradient.

Classically, both in the computer graphics and numerical analysis communities, eq. (4) would be implemented in a triangulated surface, giving place to sophisticated and elaborated algorithms even for such simple flows. We now show how to simplify this when considering implicit representations.

Recall that $\mathcal{S}$ is given as the zero level set of a function $\psi : I\!R^3 \rightarrow I\!R$, ψ is negative inside the region bounded by $\mathcal{S}$, positive outside with $\mathcal{S} \equiv \{x \in I\!R^3 : \psi(x) = 0\}$. We proceed now to redefine the above energy and compute its corresponding gradient descent flow. Let $\vec{v}$ be a generic three dimensional vector, and $P_{\vec{v}}$

the operator that projects a given three dimensional vector onto the plane orthogonal to $\vec{v}$:

$$P_{\vec{v}} := I - \frac{\vec{v} \otimes \vec{v}}{\| \vec{v} \|^2} \tag{5}$$

It is then easy to show that the harmonic energy (3) is equivalent to (see for example [30])

$$\frac{1}{2} \int_{\mathcal{S}} \| P_{\vec{N}} \nabla u \|^2 \, d\mathcal{S}, \tag{6}$$

where $\vec{N}$ is the normal to the surface $\mathcal{S}$. In other words, $\nabla_{\mathcal{S}} u = P_{\vec{N}} \nabla u$. That is, the gradient intrinsic to the surface ($\nabla_{\mathcal{S}}$) is just the projection onto the surface of the 3D Cartesian (classical) gradient ∇. We now embed this in the function ψ:

$$\begin{aligned} & \frac{1}{2} \int_{\mathcal{S}} \| \nabla_{\mathcal{S}} u \|^2 \, d\mathcal{S} = \frac{1}{2} \int_{\mathcal{S}} \| P_{\vec{N}} \nabla u \|^2 \, d\mathcal{S} \\ = & \frac{1}{2} \int_{\Omega \in \mathbb{R}^3} \| P_{\nabla \psi} \nabla u \|^2 \, \delta(\psi) \| \nabla \psi \| \, dx, \end{aligned}$$

where $\delta(\cdot)$ stands for the delta of Dirac, and all the expressions above are considered in the sense of distributions. Note that first we got rid of intrinsic derivatives by replacing $\nabla_{\mathcal{S}}$ by $P_{\vec{N}} \nabla u$ (or $P_{\nabla \psi} \nabla u$) and then replaced the intrinsic integration ($\int_{\mathcal{S}} d\mathcal{S}$) by the explicit one ($\int_{\Omega \in \mathbb{R}^3} dx$) using the delta function. Intuitively, although the energy lives in the full space, the delta function forces the penalty to be effective only on the level set of interest. The last equality includes the embedding, and it is based on the following simple facts:

1. $\nabla \psi \parallel \vec{N}$.
2. $\int_{\Omega} \delta(\psi) \| \nabla \psi \| \, dx = \int_{\mathcal{S}} d\mathcal{S} =$ surface area.

In Appendix A we show that the gradient descent of this energy is given by

$$\frac{\partial u}{\partial t} = \frac{1}{\| \nabla \psi \|} \nabla \cdot (P_{\nabla \psi} \nabla u \| \nabla \psi \|). \tag{7}$$

In other words, this equation corresponds to the intrinsic heat flow for data on an implicit surface. But all the gradients in this PDE are defined in the three dimensional Cartesian space, not in the surface $\mathcal{S}$ (this is why we need the data to be defined at least on a band around the surface). The numerical implementation is then straightforward. This is the beauty of the approach! Basically, for this equation we use a classical scheme of forward differences in time and a succession of forward and backward differences in space (see Appendix B for details). The other equations in this paper are similarly implemented. This follows techniques as those in [29]. Once again, due to the implicit representation, classic numerics are used, avoiding elaborate projections onto discrete surfaces and discretization on general meshes, e.g., [9, 17].

It is easy to show a number of important properties of this equation:

1. For any second embedding function $\phi = \phi(\psi)$, with $\phi' \neq 0$, we obtain the same gradient descent flow. Since both ψ and ϕ have to share the zero level set, and we are only interested in the flow around this zero level set, this means that the flow is (locally) independent of the embedding function.[1]
2. If ψ is the signed distance function, a very popular implicit representation of surfaces (obtained for example from the implicitation algorithms previously mentioned), the gradient descent simplifies to

$$\frac{\partial u}{\partial t} = \nabla \cdot (P_{\nabla \psi} \nabla u). \tag{8}$$

We note that we could also have derived eq. (7) directly from the harmonic maps flow

$$\frac{\partial u}{\partial t} = \Delta_{\mathcal{S}} u,$$

via the simple geometry exercise of computing $\Delta_{\mathcal{S}} u$ for $\mathcal{S}$ in implicit form. This last property is of particular significance. It basically shows how to solve general PDE's, not necessarily gradient-descent flows, for data defined on implicit surfaces. All that we need to do is to recompute the components of the PDE for implicit representations of the surface. Note that in this way, conceptually, we can re-define classical planar PDE's on implicit surfaces, making them both intrinsic to the underlying surface and defined on the whole space.

From this very simple example we have seen the key point of our approach. If the process that we want to implement comes from the minimization of an energy, we derive a PDE for the whole space by computing the gradient-descent of the whole-space-extension of that energy. Otherwise, given a planar PDE we recompute its components for an implicit representation of the surface. For instance, anisotropic diffusion can be performed on the plane by

$$\frac{\partial u}{\partial t} = \nabla \cdot \left(\frac{\nabla u}{\| \nabla u \|} \right), \tag{9}$$

which minimizes the energy $\frac{1}{2} \int_{\mathbb{R}^2} \| \nabla u \| \, dxdy$ (see [29]). If we now want to perform anisotropic diffusion of scalar data on a surface $\mathcal{S}$, we can either recompute the gradient-descent flow for an extension of the energy or just substitute in eq. (9) the corresponding expressions. Either way, we obtain the same result, the following PDE, which is valid in the Euclidean space:

$$\frac{\partial u}{\partial t} = \frac{1}{\| \nabla \psi \|} \nabla \cdot \left(\frac{P_{\nabla \psi} \nabla u}{\| P_{\nabla \psi} \nabla u \|} \| \nabla \psi \| \right). \tag{10}$$

In the following section more equations will be presented.

3 Experimental examples

We now exemplify the framework just introduced for a number of important cases. The numerical implementation used is quite simple, and requires a few lines of *C++* code. The CPU time required for the diffusion examples is of a few seconds on a PC (512Mb RAM, 1GHz) under Linux. For the texture synthesis examples, the CPU time ranges from a few minutes to one hour, depending on the pattern and parameters chosen. All the volumes used contain roughly 128^3 voxels. Note once again that due to the use of only a narrow band surrounding the zero level set, the order of the algorithmic complexity remains the same. On the other hand, the use of straightforward Cartesian numerics reduces the overall algorithmic complexity, improving accuracy and simplifying the implementation.

3.1 Diffusion of scalar images on surfaces

The use of PDE's for image enhancement has become one of the most active research areas in image processing [6]. In particular, diffusion equations are commonly used for image regularization,

[1] We thank F. Mémoli for helping with this fact.

denoising, and multiscale representations (representing the image simultaneously at several scales or levels of resolution). This started with the works in [19, 39], where the authors suggested the use of the linear heat flow (1) for this task, where u represents the image gray values (the original image is used as initial condition). As we have seen, this is the gradient-descent of (2), and the generalizations of these equations for data on the surface are given by (4) and (3) respectively. In implicit form, the heat flow on surfaces is given by (7). Figure 1 shows a simple example of image diffusion on a surface. Please note that this is *not* equivalent to performing 3D smoothing of the data and then looking to see what happened on $\mathcal{S}$. Our flow, though using extended 3D data, performs smoothing *directly* on the surface, it is an intrinsic heat flow. The complete details of the numerical implementation of this flow are given in Appendix B (this will once again show how the implementation is significantly simplified with the framework here described).

We should note before proceeding that [18] also showed how to regularize images defined on a surface. The author's approach is limited to graphs (not generic surfaces) and only applies to level set based motions. The approach is simply to project the deformation of the data on the surface onto a deformation on the plane.

3.2 Diffusion of directional data on surfaces

A particularly interesting example is obtained when we have unit vectors defined on the surface. That is, we have data of the form $u : \mathcal{S} \rightarrow S^{n-1}$. When $n = 3$ our unit vectors lie on the sphere. Particular examples of this are principal directions (or general directional fields on 3D surfaces) and chromaticity vectors (normalized RGB vectors). This is also one of the most studied cases of the theory of harmonic maps due to its physical relationship with liquid crystals, and it was introduced in [32] for the regularization of directional data, unit vectors, on the plane. This framework of harmonic maps was used in computer graphics for texture mapping and surface parameterization, as pointed out earlier.

We still want to minimize an energy of the form

$$\int_{\mathcal{S}} \parallel \nabla_{\mathcal{S}} u \parallel^p d\mathcal{S},$$

though in this case $\nabla_{\mathcal{S}}$ is the vectorial gradient and the minimizer is restricted to be a unit vector. It is easy to show, e.g., [4, 31], that the gradient descent of this energy is given by the coupled system of PDE's

$$\frac{\partial u_i}{\partial t} = \mathrm{div}_{\mathcal{S}} \left(\parallel \nabla_{\mathcal{S}} u \parallel^{p-2} \nabla_{\mathcal{S}} u_i \right) + u_i \parallel \nabla_{\mathcal{S}} u \parallel^p, \;\; 1 \leq i \leq n.$$

This flow guarantees that the initial unit vector $u(x, y, z, 0)$ remains a unit vector $u(x, y, z, t)$ all the time, thereby providing an equation for isotropic ($p = 2$) and anisotropic ($p = 1$) diffusion and regularization of unit vectors on a surface.

We can now proceed as before, and embed the surface $\mathcal{S}$ into the zero level-set of ψ, obtaining the following gradient descent flows:

$$\frac{\partial u_i}{\partial t} = \frac{1}{\parallel \nabla \psi \parallel} \nabla \cdot \left(\frac{P_{\nabla \psi} \nabla u_i}{\parallel P_{\nabla \psi} \nabla u \parallel^{2-p}} \parallel \nabla \psi \parallel \right) + u_i \parallel P_{\nabla \psi} \nabla u \parallel^p . \tag{11}$$

Note once again that although the regularization is done intrinsically on the surface, this equation only contains Cartesian gradients. An example of this flow for anisotropic diffusion of principal direction vectors is given in Figure 2. On the left, we see the surface of a bunny with its correspondent vector field for the major principal direction. Any irregularity on the surface produces a noticeable alteration of this field, as can be seen in the details *a* and *b*. In the details *a'* and *b'*, we see the result of applying the flow (11). Once again, the implementation of this flow with our framework is straightforward, while it would require very sophisticated techniques on triangulated surfaces (techniques that, in addition, are not supported by theoretical results).

Following also the work [32] for color images defined on the plane, we show in Figure 3 how to denoise a color image painted on an implicit surface. The basic idea is to normalize the RGB vector (a three dimensional vector) to a unit vector representing the chroma, and diffuse this unit vector with the harmonic maps flow (11).[2] The corresponding magnitude, representing the brightness, is smoothed separately via scalar diffusion flows as those presented before (e.g., the intrinsic heat flow or the intrinsic anisotropic heat flow). That is, we have to regularize a map onto S^2 (the chroma) and another one onto $\mathbb{R}$ (the brightness).

3.3 Pattern formation on surfaces via reaction-diffusion flows

The use of reaction-diffusion equations for texture synthesis became very popular in computer graphics following the works of Turk [37] and Witkin and Kass [41]. These works follow original ideas by Turing [36], who showed how reaction diffusion equations can be used to generate patterns. The basic idea in these models is to have a number of "chemicals" that diffuse at different rates and that react with each other. The pattern is then synthesized by assigning a brightness value to the concentration of one of the chemicals. The authors in [37, 41] used their equations for planar textures and textures on triangulated surfaces. By using the framework here described, we can simply create textures on (implicit/implicitized) surfaces, without the elaborated schemes developed in those papers.[3]

Assuming a simple isotropic model with just two chemicals u_1 and u_2, we have

$$\frac{\partial u_1}{\partial t} = F(u_1, u_2) + D_1 \Delta u_1,$$

$$\frac{\partial u_2}{\partial t} = G(u_1, u_2) + D_2 \Delta u_1,$$

where D_1 and D_2 are two constants representing the diffusion rates and F and G are the functions that model the reaction.

Introducing our framework, if u_1 and u_2 are defined on a surface $\mathcal{S}$ implicitly represented as the zero level set of ψ we have

$$\frac{\partial u_1}{\partial t} = F(u_1, u_2) + D_1 \frac{1}{\parallel \nabla \psi \parallel} \nabla \cdot (P_{\nabla \psi} \nabla u_1 \parallel \nabla \psi \parallel), \tag{12}$$

$$\frac{\partial u_2}{\partial t} = G(u_1, u_2) + D_2 \frac{1}{\parallel \nabla \psi \parallel} \nabla \cdot (P_{\nabla \psi} \nabla u_2 \parallel \nabla \psi \parallel). \tag{13}$$

For simple isotropic patterns, Turk [37] selected

$$F(u_1, u_2) = s(16 - u_1 u_2),$$

$$G(u_1, u_2) = s(u_1 u_2 - u_2 - \beta),$$

where s is a constant and β is a random function representing irregularities in the chemical concentration. Examples of this, for implicit surfaces, are given in Figure 4 (the coupled PDE's shown above are run until steady state is achieved). To simulate anisotropic textures, instead of using additional chemicals as in [37], we use

[2] We re-normalize at every discrete step of the numerical evolution to address deviations from the unit norm due to numerical errors [8]. We could also extend the framework in [1] and apply it to our equations.

[3] Note that this is not the scheme proposed in [25], where the texture is created in the full 3D space. Here, the texture is created via reaction-diffusion flows intrinsic to the surface, just the implementation is on the embedding 3D space.

anisotropic diffusion, as suggested in [41]. For this purpose, we replace eq. (12) with:

$$\frac{\partial u_1}{\partial t} = F(u_1, u_2) + D_1 \frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot ((\overrightarrow{d} \cdot P_{\nabla\psi} \nabla u_1)\overrightarrow{d} \parallel \nabla\psi \parallel), \tag{14}$$

where $\overrightarrow{d}$ is a vector field tangent to the surface, e.g., the field of the major principal direction (which for our examples has been also accurately computed directly on the implicit surface, using the technique proposed in [22]). Note how this particular selection of the anisotropic reaction-diffusion flow direction provides a texture that helps on the shape perception of the object. Additional patterns can be obtained with different combinations of the reaction and diffusion parts of the flow.

3.4 Flow visualization on 3D surfaces

Inspired by the work on line integral convolution [5] and that on anisotropic diffusion [27], the authors of [10] suggested to use anisotropic diffusion to visualize flows in 2D and 3D. The basic idea is, starting from a random image, anisotropically diffuse it in the directions dictated by the flow field. The authors presented very nice results both in 2D (flows on the plane) and 3D (flows on a surface), but once again using triangulated surfaces which introduce many computational difficulties. In a straightforward fashion we can compute these anisotropic diffusion equations on the implicit surfaces with the framework here introduced, and some results are presented in Figure 5. Note the complicated topology and how both the inside and outside parts of the surfaces are easily handled with our implicit approach. Also note that, when we choose the vector field to be that of one of the principal directions, the result emphasizes the surface shape.

4 Concluding remarks

In this paper, we have introduced a novel framework for solving variational problems and PDE's for data defined on surfaces. The technique borrows ideas from the level set theory and the theory of harmonic maps. The surface is embedded in a higher dimensional function, and the Euler-Lagrange flow or PDE is solved in the Cartesian coordinate system of this embedding function. The equations are intrinsic to the implicit surface, following the general formulations in harmonic map theory. With this framework we enjoy accuracy, robustness, and simplicity, as expected from the computation of differential characteristics on iso-surfaces (implicit surfaces). In addition to presenting the general approach, we have exemplified it with equations arising in image processing and computer graphics. We are currently investigating other applications of this framework, e.g, image inpainting on surfaces [2], inverse problems as those in [15], and texture mapping for implicit surfaces following [12].

Acknowledgment

We thank Facundo Mémoli for interesting conversations during this work. The implicit data provided by S. Betelu and H. Zhao. This work was partially supported by grants from the Office of Naval Research ONR-N00014-97-1-0509 and ONR-N00014-97-1-0027, the Office of Naval Research Young Investigator Award, the Presidential Early Career Awards for Scientists and Engineers (PECASE), a National Science Foundation CAREER Award, the National Science Foundation Learning and Intelligent Systems Program (LIS), NSF-DMS-9706827, ARO-DAAG-55-98-1-0323 and IIE-Uruguay.

Appendix A: Heat flow on implicit surfaces

Considering

$$E(u) := \frac{1}{2} \int_{\Omega \in I\!R^3} \parallel P_{\nabla\psi} \nabla u \parallel^2 \delta(\psi) \parallel \nabla\psi \parallel dx,$$

and μ a perturbation of u,

$$\begin{aligned}
\tfrac{d}{dt} \mid_{t=0} E(u + t\mu) &= \int_\Omega (P_{\nabla\psi}\nabla u \cdot P_{\nabla\psi}\nabla\mu)\delta(\psi) \parallel \nabla\psi \parallel dx \\
&= \int_\Omega \left(P_{\nabla\psi}\nabla u \cdot \left(\nabla\mu - \frac{\nabla\psi \cdot \nabla\mu}{\parallel \nabla\psi \parallel^2} \nabla\psi \right) \right) \delta(\psi) \parallel \nabla\psi \parallel dx \\
&= \int_\Omega (P_{\nabla\psi}\nabla u \cdot \nabla\mu)\delta(\psi) \parallel \nabla\psi \parallel dx \\
&\quad - \int_\Omega (P_{\nabla\psi}\nabla u \cdot \nabla\psi) \frac{\nabla\psi \cdot \nabla\mu}{\parallel \nabla\psi \parallel^2} \delta(\psi) \parallel \nabla\psi \parallel dx \\
&= \int_\Omega (P_{\nabla\psi}\nabla u \cdot \nabla\mu)\delta(\psi) \parallel \nabla\psi \parallel dx \\
&= -\int_\Omega \nabla \cdot (P_{\nabla\psi}\nabla u \delta(\psi) \parallel \nabla\psi \parallel)\mu dx \\
&= -\int_\Omega \nabla \cdot (P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel)\delta(\psi)\mu dx \\
&\quad - \int_\Omega (P_{\nabla\psi}\nabla u \cdot \nabla\psi)\delta'(\psi) \parallel \nabla\psi \parallel \mu dx \\
&= -\int_\Omega \nabla \cdot (P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel)\delta(\psi)\mu dx \\
&= -\int_{\mathcal{S} \equiv \{\psi = 0\}} \frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot (P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel)\mu d\mathcal{S}.
\end{aligned}$$

Since the above has to be zero for all μ, we conclude that at the zero level set of ψ,

$$\frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot (P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel) = 0,$$

and we make a natural extension to the whole domain Ω by considering this to hold on it.[4] We then obtain that the gradient descent for the "implicit harmonic energy" is given by

$$\frac{\partial u}{\partial t} = \frac{1}{\parallel \nabla\psi \parallel} \nabla \cdot (P_{\nabla\psi}\nabla u \parallel \nabla\psi \parallel). \tag{15}$$

Appendix B: Numerical implementation of the heat flow on implicit surfaces

We now provide details on the numerical implementation of the intrinsic heat flow on implicit surfaces. All other equations reported in this paper are similarly implemented. Recall that all equations are on Cartesian grids, thereby permitting the use of classical numerics. We work on a cubic grid ($\Delta x = \Delta y = \Delta z = 1$), using an explicit scheme, where we compute the value of $u^n_{i,j,k} = u(i\Delta x, j\Delta y, k\Delta z, n\Delta t)$ based only in previous values ($t = (n-1)\Delta t$) of its neighbors, i.e., forward time differences.

[4] We have assumed that $\parallel \nabla\psi \parallel \neq 0$, at least on a band surrounding the zero level set. This assumption is valid since we can make the embedding function to be a distance function ($\parallel \nabla\psi \parallel = 1$), or simply multiply ψ by another function that guarantees that the zero-level set $\mathcal{S}$ is preserved and that the gradient of the new embedding function is not zero.

Firstly, we compute the 3D gradient of u using forward differences:

$\overrightarrow{v}^n_{i,j,k} = \nabla_+ u^n_{i,j,k} = (u^n_{i+1,j,k} - u^n_{i,j,k}, u^n_{i,j+1,k} - u^n_{i,j,k}, u^n_{i,j,k+1} - u^n_{i,j,k})$.

We compute the vector $\overrightarrow{N}(i,j,k)$, which gives the direction of the (outward) normal to the isosurface of ψ at the point (i,j,k):

$\overrightarrow{N}_{i,j,k} = \nabla\psi_{i,j,k} = \frac{1}{2}(\psi_{i+1,j,k} - \psi_{i-1,j,k}, \psi_{i,j+1,k} - \psi_{i,j-1,k}, \psi_{i,j,k+1} - \psi_{i,j,k-1})$.

Here we have used central differences. Note that, since ψ is fixed, $\overrightarrow{N}(i,j,k)$ does not change in time and we need only to compute it once. Its norm is: $\| \overrightarrow{N}_{i,j,k} \| = (\sum_{m=1}^{3} (N_{i,j,k}[m])^2)^{\frac{1}{2}}$. The square brackets denote the components of the vector. Then we compute the intrinsic gradient, i.e., we project ∇u onto the plane normal to $\overrightarrow{N}$:

$(P_{\overrightarrow{N}} \overrightarrow{v})^n_{i,j,k} = \overrightarrow{v}^n_{i,j,k} - (\frac{\sum_{m=1}^{3} N_{i,j,k}[m] \cdot v_{i,j,k}[m]}{\| \overrightarrow{N}_{i,j,k} \|^2}) \overrightarrow{N}_{i,j,k}$.

Finally, we use a backward-differences implementation of the divergence:

$\nabla_- \overrightarrow{w}_{i,j,k} = w_{i,j,k}[1] - w_{i-1,j,k}[1] + w_{i,j,k}[2] - w_{i,j-1,k}[2] + w_{i,j,k}[3] - w_{i,j,k-1}[3]$.

With forward time differences, the numerical implementation of the heat flow on implicit surfaces (eq. (7)) is then:

$u^{n+1}_{i,j,k} = u^n_{i,j,k} + \Delta t (\frac{1}{\| \overrightarrow{N}_{i,j,k} \|} \nabla_- \cdot ((P_{\overrightarrow{N}} \overrightarrow{v})^n_{i,j,k} \| \overrightarrow{N}_{i,j,k} \|))$.

If the embedding function is a signed distance function, obtaining eq. (8), this expression for the numerical implementation of the intrinsic heat flow is simplified even further, making it virtually trivial.

References

[1] F. Alouges, "An energy decreasing algorithm for harmonic maps," in J.M. Coron *et al.*, Editors, *Nematics*, Nato ASI Series, Kluwer Academic Publishers, Netherlands, pp. 1-13, 1991.

[2] M. Bertalmío, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," *Computer Graphics (SIGGRAPH)*, pp. 417-424, New Orleans, July 2000.

[3] J. Bloomenthal, *Introduction to Implicit Surfaces*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997.

[4] H. Brezis, J. M. Coron, and E. H. Lieb, "Harmonic maps with defects," *Communications in Mathematical Physics* **107**, pp. 649-705, 1986.

[5] B. Cabral and C. Leedom. "Imaging vector fields using line integral convolution," *ACM Computer Graphics (SIGGRAPH '93)* **27:4**, pp. 263-272, 1993.

[6] V. Caselles, J. M. Morel, G. Sapiro, and A. Tannenbaum, Editors, "Special Issue on Partial Differential Equations and Geometry-Driven Diffusion in Image Processing and Analysis," *IEEE Trans. Image Processing* **7**, March 1998.

[7] S. Chen, B. Merriman, S. Osher, and P. Smereka, "A simple level set method for solving Stefan problems," *Journal of Computational Physics* **135**, pp. 8, 1995.

[8] R. Cohen, R. M. Hardt, D. Kinderlehrer, S. Y. Lin, and M. Luskin, "Minimum energy configurations for liquid crystals: Computational results," in J. L. Ericksen and D. Kinderlehrer, Editors, *Theory and Applications of Liquid Crystals*, pp. 99-121, IMA Volumes in Mathematics and its Applications, Springer-Verlag, New York, 1987.

[9] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Discrete differential-geometry operators in nD," *Multi-res modeling group TR*, Caltech, September 2000 (obtained from www.multires.caltech.edu).

[10] U. Diewald, T. Preufer, and M. Rumpf, "Anisotropic diffusion in vector field visualization on Euclidean domains and surfaces," *IEEE Trans. Visualization and Computer Graphics* **6**, pp. 139-149, 2000.

[11] J. Dorsey and P. Hanrahan, "Digital materials and virtual weathering," *Scientific American* **282:2**, pp. 46-53, 2000.

[12] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. "Multi-resolution analysis of arbitrary meshes," *Computer Graphics (SIGGRAPH '95 Proceedings)*, pp. 173-182, 1995.

[13] M. Eck and H. Hoppe, "Automatic reconstruction of B-spline surfaces of arbitrary topological type," *Computer Graphics*, 1996.

[14] J. Eells and L. Lemarie, "A report on harmonic maps," *Bull. London Math. Soc.* **10:1**, pp. 1-68, 1978.

[15] O. Faugeras, F. Clément, R. Deriche, R. Keriven, T. Papadopoulo, J. Gomes, G. Hermosillo, P. Kornprobst, D. Lingrad, J. Roberts, T. Viéville, F. Devernay, "The inverse EEG and MEG problems: The adjoint state approach I: The continuous case," *INRIA Research Report* **3673**, June 1999.

[16] S. F. Frisken, R. N. Perry, A. Rockwood, and T. Jones, "Adaptively sampled fields: A general representation of shape for computer graphics," *Computer Graphics (SIGGRAPH)*, New Orleans, July 2000.

[17] G. Huiskamp, "Difference formulas for the surface Laplacian on a triangulated surface," *Journal of Computational Physics* **95**, pp. 477-496, 1991.

[18] R. Kimmel, "Intrinsic scale space for images on surfaces: The geodesic curvature flow," *Graphical Models and Image Processing* **59**, pp. 365-372, 1997.

[19] J. J. Koenderink, "The structure of images," *Biological Cybernetics* **50**, pp. 363-370, 1984.

[20] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," *Computer Graphics*, pp. 313-324, 1996.

[21] S. Mauch, "Closest point transform," www.ama.caltech.edu/∼seanm/software/cpt/cpt.html.

[22] O. Monga, S. Benayoun and O. Faugeras, "From partial derivatives of 3D density images to ridge lines," *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 354-359, 1992.

[23] H. P. Moreton and C. H. Séquin, "Functional minimization or fair surface design," *Computer Graphics (SIGGRAPH)*, pp. 167-176, 1992.

[24] S. J. Osher and J. A. Sethian, "Fronts propagation with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics* **79**, pp. 12-49, 1988.

[25] K. Perlin, "An image synthesizer," *Computer Graphics* **19**, pp. 287-296, 1985.

[26] D. Peng, B. Merriman, S. Osher, H. Zhao, M. Kang, "A PDE-based fast local level set method," *Journal of Computational Physics* **155**, pp. 410-438, 1999.

[27] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern. Anal. Machine Intell.* **12**, pp. 629-639, 1990.

[28] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped textures," *ACM Computer Graphics (SIGGRAPH)*, New Orleans, July 2000.

[29] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D* **60**, pp. 259-268, 1992.

[30] L. Simon, *Lectures on Geometric Measure Theory*, Australian National University, Australia, 1984.

[31] M. Struwe, "On the evolution of harmonic mappings of Riemannian surfaces," *Comment. Math. Helvetici* **60**, pp. 558-581, 1985.

[32] B. Tang, G. Sapiro, and V. Caselles, "Diffusion of general data on non-flat manifolds via harmonic maps theory: The direction diffusion case," *Int. Journal Computer Vision* **36:2**, pp. 149-161, February 2000.

[33] G. Taubin, "Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation," *IEEE Trans. PAMI* **13:11**, pp. 1115-1138, 1991.

Figure 1: Intrinsic isotropic diffusion. Left: original image. Middle: after 60 diffusion steps. Right: after 160 diffusion steps.

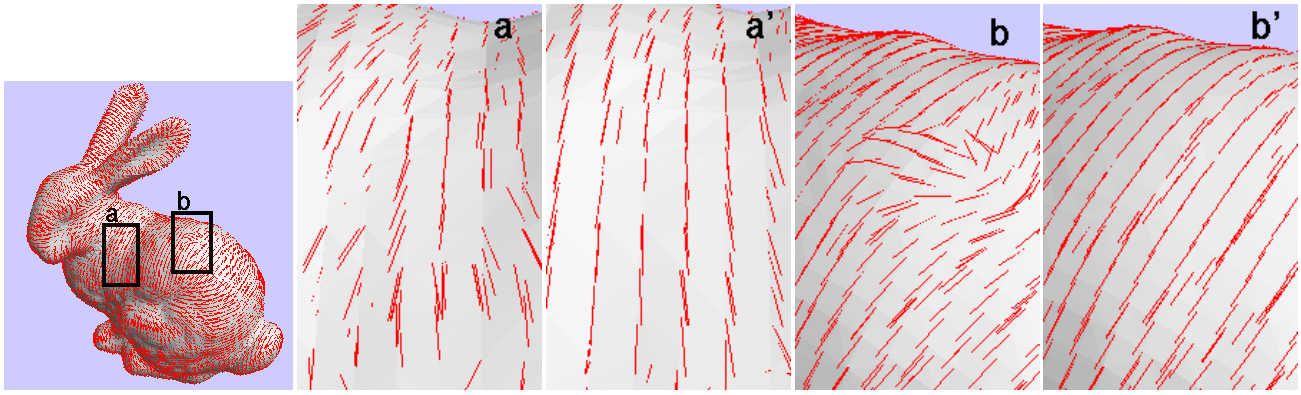


Figure 2: Intrinsic vector field regularization. Left: original field of major principal direction of the surface. Details *a* and *b*: original field. Details *a'* and *b'*: after anisotropic regularization.

[34] G. Taubin, "Estimating the tensor of curvature of a surface from a polyhedral approximation," *IEEE International Conference Computer Vision*, pp. 902-907, Boston, MA, 1995.

[35] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control* **40** pp. 1528-1538, 1995.

[36] A. Turing, "The chemical basis of morphogenesis," *Philosophical Transactions of the Royal Society B* **237**, pp. 37-72, 1952.

[37] G. Turk, "Generating textures on arbitrary surfaces using reaction-diffusion," *Computer Graphics* **25:4**, pp. 289-298, , July 1991.

[38] G. Winkenbach and D. H. Salesin, "Rendering parametric surfaces in pen and ink," *Computer Graphics (SIGGRAPH 96)*, pp. 469-476, 1996.

[39] A. P. Witkin, "Scale-space filtering," *Int. Joint. Conf. Artificial Intelligence* **2**, pp. 1019-1021, 1983.

[40] A. Witkin and P. Heckbert, "Using particles to sample and control implicit surfaces," *Computer Graphics (SIGGRAPH)*, pp. 269-278, 1994.

[41] A. Witkin and M. Kass, "Reaction-diffusion textures," *Computer Graphics (SIGGRAPH)* **25:4**, pp. 299-308, July 1991.

[42] G. Yngve and G. Turk, "Creating smooth implicit surfaces from polygonal meshes," *Technical Report GIT-GVU-99-42, Graphics, Visualization, and Usability Center. Georgia Institute of Technology*, 1999 (obtained from www.cc.gatech.edu/gvu/geometry/publications.html).

[43] D. Zhang and M. Hebert, "Harmonic maps and their applications in surface matching," *Proc. CVPR '99*, Colorado, June 1999.

[44] H. K. Zhao, T. Chan, B. Merriman, and S. Osher, "A variational level set approach to multi-phase motion," *J. of Computational Physics* **127**, pp. 179-195, 1996.

[45] H. Zhao, S. Osher, B. Merriman, and M. Kang, "Implicit, nonparametric shape reconstruction from unorganized points using a variational level set method," *Comp. Vision and Image Understanding* **80**, pp. 295-314, 2000.

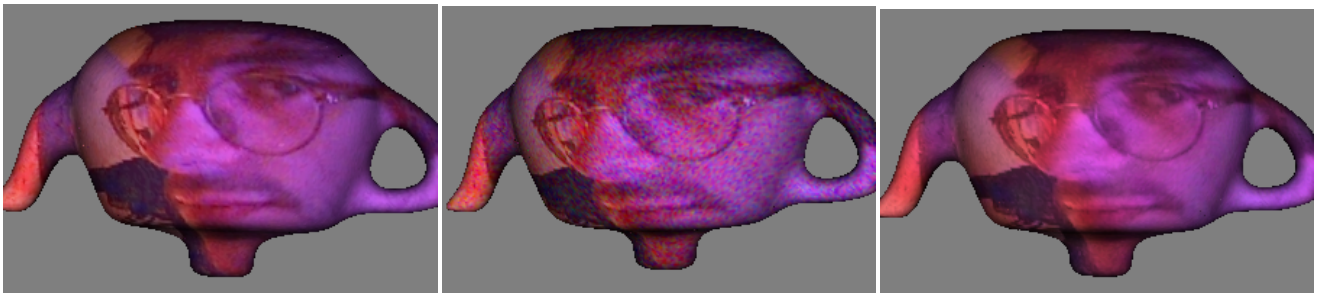


Figure 3: Intrinsic vector field regularization. Left: original color image. Middle: heavy noise has been added to the 3 color channels. Right: color image reconstructed after 20 steps of anisotropic diffusion of the chroma vectors.

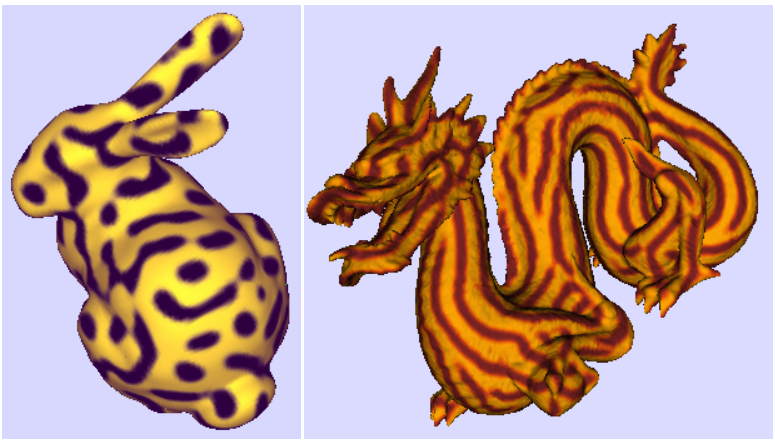


Figure 4: Texture synthesis via intrinsic reaction-diffusion flows on implicit surfaces. Left: isotropic. Right: anisotropic. Pseudo-color representation of scalar data is used. The numerical values used in the computations were $D_1 = 1.0, D_2 = 0.0625, s = 0.025, \beta = 12.0 \pm 0.1, u_1(0) = u_2(0) = 4.0$.

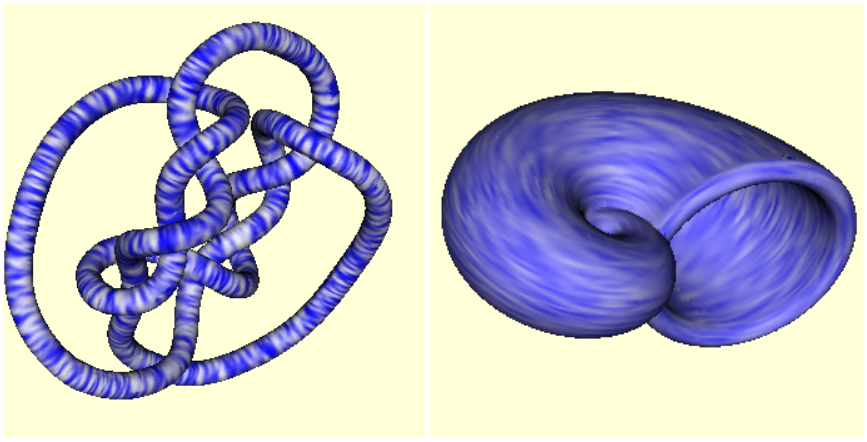


Figure 5: Flow visualization on implicit 3D surfaces via intrinsic anisotropic diffusion flows. Left: flow aligned with the major principal direction of the surface. Right: flow aligned with the minor principal direction of the surface. Pseudo-color representation of scalar data is used.

A Method For Denoising Textured Surfaces

S. Betelu,
Institute for Mathematics and its Applications,
University of Minnesota

A. Tannenbaum
Dept. of Electrical and Computer Engineering
Georgia Institute of Technology,

G. Sapiro
Dept. of Electrical and Computer Engineering
University of Minnesota

May 2, 2001

Abstract

In this note, we present a simple method to denoise triangulated and implicit surfaces in a manner which preserves the 3D shape texture. The technique is based upon the synthesis of partial differential equations (PDE's), implicit surfaces, and Wiener filtering. The basic idea is to apply a computationally efficient local Wiener filter to an implicit representation of the surface. Such a representation can be directly given as the algorithm input or explicitly obtained via partial differential equation based implicitation techniques applied to the triangulated data. The proposed method has a computational complexity $O(N \log N)$.

1 INTRODUCTION

In this note, we consider the problem of the restoration of surfaces that have been corrupted by noise. The given surface S may have a complex topology (with handles, holes, unconnected parts, etc.) and have *shape texture*, i.e., a 3D locally periodic modulation on the shape itself, as illustrated in Fig. 2-a. The goal is to remove the noise while preserving the shape "texture." This is an important problem, since in every practical application concerning real imagery derived from various modalities, the measured data may be corrupted by noise, which has to be removed while preserving the features of the given shape.

The problem of surface denoising (or surface smoothing and surface fairing) has been considerably studied in recent years using a number of different approaches. These include the direct processing of the triangulation surface, e.g.,

[2, 3, 20], as well as the processing of an implicit representation of the surface, e.g., [15, 18] and references therein.

More specifically, in [3], estimates for the local curvatures of the surface are developed starting from a polyhedral surface. Then the surface is evolved (deformed) in the direction of the normal as a function of the local curvatures in order to remove the noise. Work has also been done to find the best velocity field as a function of the curvatures and other local quantities in order to preserve the features of the original shape. In [2], a velocity field is constructed using the signs of the principal curvatures in order to preserve corners, while in [1], the discrete surface is directly evolved with an anisotropic geometric evolution equation. Hence the approach considered in these works is based on deforming the triangulated surfaces via certain geometric partial differential equations (PDE). These methods can also deal with implicit surfaces following the implicitation process as proposed below. We should note that curvature based deformations of implicit surfaces are common in the mathematical physics literature, e.g., [12] and the references therein.

An elegant signal processing approach for surface smoothing is proposed in [20]. The classical discrete Fourier analysis and filter design is generalized to functions defined on polyhedral surfaces, and a low pass filter is designed and implemented. In this approach, the vertices of a polyhedral surface are moved without changing the connectivity of the faces or the number of faces or vertices. Also, the algorithm avoids surface shrinking. Another recent method that applies filters directly to triangulated meshes is developed in [5].

The use of implicit functions for surface deformations has been advocated in [12], where in particular mean curvature type flows are implemented. It is argued that the implementation of curvature based surface deformations is more robust on implicit surfaces than on triangulated ones. Other examples of denoising implicit surfaces include the work on min-max in [18] and the work on anisotropic diffusion recently reported in [15]. In this last work, the corresponding anisotropic diffusion (a parabolic PDE) is designed to preserve edges and corners.

In our paper, we are particularly interested in surfaces with texture on the shape. Thus, in this case it is natural to look for a method that uses directly the information of the local modes of the surface texture. This suggests the use of the fast Fourier transform (FFT) and Wiener filtering. We now outline our simple approach to the denoising of textured surfaces.

1.1 OUTLINE OF METHODS AND CONTRIBUTIONS

Our simple proposed approach is based on a natural algorithmic synthesis of implicit surface representations, Fourier transforms and PDE's. More specifically, we propose the use of Wiener filtering on implicit surface representations. It is well-known that the Wiener filter, when applied locally to digital *textured* images, provides outstanding performance in removing noise while preserving fine details and the texture [8, 13, 22, 6]. An example of this is given in Figure 1 (the details on the exact implementation of the filter in this case are the 2D

analogue of the 3D method introduced below).

We now give some details about our application of the Wiener filter to noisy surfaces. Let the surface be given in implicit form,

$$S = \{x \in \mathbf{R}^3 : \phi(x) = 0\} \tag{1}$$

or as a set of N_T triangles

$$S = \bigcup_{i=1}^{N_T} T_i. \tag{2}$$

For implicit surfaces, we apply locally a simplified Wiener filter to the implicit function ϕ, and then reconstruct the surface $\phi_s(x) = 0$, for example via the marching cubes algorithm [11]. We shall describe this procedure in Section 2.

For triangulated surfaces, our method, which follows a growing trend in the computer vision and computer graphics communities, is to first compute the implicit (signed) distance function ϕ in a $3D$ band that surrounds the surface. That is, we first compute

$$|\nabla \phi| = 1, \qquad \phi(x) = 0 \quad \text{for} \quad x \in S, \tag{3}$$

and then smooth ϕ with the local Wiener filter to obtain ϕ_s, and finally reconstruct the surface defined by $\phi_s(x) = 0$. We briefly describe this procedure in Section 3. The solutions of Eq. 3 can be computed very efficiently with a number of recently proposed techniques, e.g., [7, 10, 16, 17, 21, 23, 25]. Further, the local Wiener filter be efficiently computed with the FFT. The resulting algorithm is thus $O(N \log N)$, where N is the number of points on which ϕ is computed.

One attractive feature of the method that we propose in this work is its *simplicity.* It is based on two straightforward algorithms - Wiener filtering and fast implicitation methods - that are very well understood and have been extensively studied and applied over many years in the engineering and scientific computing communities. These techniques are efficient, robust and easy to program.

An important advantage of our approach over curvature flows [18, 3, 2] and second order PDE's [1, 15] is that the Wiener filter detects the local periodicity of the surface. Thus, for example, if ripples or a periodic array of holes are present in the initial surface, they will be preserved. We show an illustrative example of this property in Section 4. In contrast, curvature flows are much less sensitive to periodic structures because the restoration uses less information (namely, local curvatures or second derivatives, instead of the local Fourier transform over larger length scales). Moreover, the Wiener filter removes noise according to a noise model, which differentiates random noise from signal. However, typical curvature flows do not discriminate between noise and high frequency features of the image.

The algorithm may be extended in a very straightforward way to include more sophisticated noise statistics, as pointed out in Section 2. That enlarges the flexibility of the method enormously. In particular, if the noise spectrum is

known *a priori*, its inclusion on the algorithm will not degrade the computational efficiency.

2 SIMPLIFIED WIENER FILTER

In this section, we describe how to smooth the distance function (or any other implicit representation of the surface) $\phi : \mathbf{R}^3 \rightarrow \mathbf{R}$ with the local Wiener filter. Accordingly, we assume that ϕ contains an additive noise component $n(x)$

$$\phi(x) = \phi_0(x) + n(x), \tag{4}$$

with $x \in \mathbf{R}^3$ and $\phi_0(x)$ is the uncorrupted scalar. Let us denote by $\Phi(k)$, $\Phi_0(k)$ and $N(k)$ the respective Fourier transforms with $k \in \mathbf{R}^3$. We search for a linear estimate Φ_a for the uncorrupted signal given the corrupted signal Φ. This is done in the following way:

$$\Phi_a(k) = W(k)\Phi(k). \tag{5}$$

In the least squares sense, the distance between Φ_a and Φ_0 is

$$E = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |\phi_a(x) - \phi_0(x)|^2 dx^3 \tag{6}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |\Phi_a(k) - \Phi_0(k)|^2 dk^3.$$

It is well-known that

$$W(k) = \frac{|\Phi_0(k)|^2}{|\Phi_0(k)|^2 + |N(k)|^2} \tag{7}$$

minimizes the above integral, provided the noise is stationary, non-correlated with the signal and has zero mean.

In order to use this optimal linear estimator, we are confronted with the practical problem of finding estimates for the unknown $|\Phi_0(k)|^2$ and $|N(k)|^2$. Here, we shall make a drastic simplification: we will assume that

$$|\Phi_0(k)| \sim |\Phi(k)| \qquad \text{and} \qquad |N(k)| = N_0 \tag{8}$$

where N_0 is a free parameter which must be given by the user. Even for this simplified model, very good denoising results are obtained as we will report in the experimental section. If for some particular problem the statistics of the noise and its correlation with the signal are known, they may be used to estimate $N(k)$ more accurately [8].

It is well-known that in order to avoid blurring and better preserve signal details and texture, the Wiener filter must be applied locally for every point x in the domain [22, 6]. We adopt this and optimize the method to increase the speed of processing by doing a further approximation: We evaluate the filter on a set of overlapping cubes of $M \times M \times M$ gridpoints, where M is a parameter

given by the user. It is better to use overlapping cubes rather than separate cubes to avoid spurious discontinuities between them.

To recap, for noisy implicit surfaces we propose to apply a local Wiener filter, with a moving overlapping window and with a simplified noise and signal model. To conclude this section, we describe the algorithm in pseudocode format:

Input: Let the input data, ϕ, be stored in a three dimensional array of numbers ϕ_{ijk} with $0 \leq i < N_x$, $0 \leq j < N_y$ and $0 \leq k < N_z$.

1- Initialize $\phi_{s\,i,j,k} = 0$, $0 \leq i < N_x$, $0 \leq j < N_y$ and $0 \leq k < N_z$.
2- For ($0 \leq I < 2N_x/M$, $0 \leq J < 2N_y/M$ and $0 \leq K < 2N_z/M$) **do**

- Compute the discrete local Fourier transform

$$\Phi_{k_x k_y k_z} = \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} \sum_{z=0}^{M-1}$$

$$\phi_{x+I\frac{M}{2},y+J\frac{M}{2},z+K\frac{M}{2}} \exp 2\pi i(xk_x + yk_y + zk_z)$$

  using the 3D fast Fourier transform algorithm, for $0 \leq k_x < M$, $0 \leq k_y < M$ and $0 \leq k_z < M$.

- Compute the Wiener filter to the coefficients in the following way:

$$\Phi'_{k_x k_y k_z} = \begin{cases} \Phi_{k_x k_y k_z} \frac{\Phi^2_{k_x k_y k_z}}{M^3 N_0^2 + \Phi^2_{k_x k_y k_z}}, & |\mathbf{k}|^2 > 3 \\ \Phi_{k_x k_y k_z} & |\mathbf{k}|^2 \leq 3 \end{cases}$$

  The last line implies that the average value of $\Phi(k)$ and its lowest modes will be conserved. Numerical experiments show that this helps to reduce spurious thinning on the shape.

- Compute the inverse Fourier transform for each cube and store it in the array ϕ', where

$$\phi'_{x+I\frac{M}{2},y+J\frac{M}{2},z+K\frac{M}{2}} =$$

$$\frac{1}{M^3} \sum_{k_x=0}^{M-1} \sum_{k_y=0}^{M-1} \sum_{k_z=0}^{M-1} \Phi'_{k_x k_y k_z} \exp -2\pi i(xk_x + yk_y + zk_z)$$

- The smoothed ϕ_s is computed as the following weighted sum over all the cubes:
  For ($0 \leq x < M$, $0 \leq y < M$, $0 \leq z < M$) **do**

$$\phi_{s\,x+I\frac{M}{2},y+J\frac{M}{2},z+K\frac{M}{2}} = \phi_{s\,x+I\frac{M}{2},y+J\frac{M}{2},z+K\frac{M}{2}} +$$

$$\frac{8}{M^3} \left(\frac{M}{2} - \left|x - \frac{M}{2}\right|\right) \left(\frac{M}{2} - \left|y - \frac{M}{2}\right|\right)$$

$$\left(\frac{M}{2} - \left|z - \frac{M}{2}\right|\right) \phi'_{x+I\frac{M}{2}, y+J\frac{M}{2}, z+K\frac{M}{2}}$$

end for

end for

The output of the algorithm is the vector with the smoothed ϕ_s. The temporal complexity of the method is $O(N_x N_y N_z \log M)$.

3 CONSTRUCTION OF THE DISTANCE FUNCTION FROM A TRIANGULATED SURFACE

When the data is presented as a triangulated surface with N_T triangles, we must construct the distance function or a related implicit representation on the space around the given surface. There are several different ways to do so [4, 9, 10, 19, 24, 14, 25]. In the examples presented below we use the *fast marching* approach [7, 16, 21]. This gives a powerful implementation of the Eikonal equation, which is very easy to code.

We implement a 3-step procedure:

A) Discretize the rectangular volumetric box of dimensions (L_x, L_y, L_z) around the surface with a uniform grid x_{ijk} with $1 \leq i \leq N_x$, $1 \leq j \leq N_y$, and $1 \leq k \leq N_z$. In this grid, we shall compute the distance function ϕ_{ijk}.

B) Compute the exact distance to the surface in a band of one grid point around the surface. This is computed as the minimum distance to the triangles T_i. The sign of ϕ can be computed by assigning positive values on the side where the normals of the triangles lie, and negative values on the other side.

C) For the remaining points of the grid, we compute the distance function ϕ_{ijk} by solving the finite differences approximation of Eq. (3)

$$\begin{aligned} &\max(D^{-x}_{ijk}\phi, -D^{x}_{ijk}\phi, 0)^2 + \\ &\max(D^{-y}_{ijk}\phi, -D^{y}_{ijk}\phi, 0)^2 + \\ &\max(D^{-z}_{ijk}\phi, -D^{z}_{ijk}\phi, 0)^2 = 1. \end{aligned} \tag{9}$$

This Godunov scheme guarantees that the numerical solution approximates at first order the viscosity solution of Eq. (3). We solve this system of equations with the fast marching algorithm first proposed in [21] following an optimal control approach and independently derived in [7, 16, 17] following an upwind-numerical approach. This technique has an optimal temporal complexity of $O(N \log N)$ where N is the number of points where ϕ is computed. It is not necessary to compute it in the entire grid, since its values are only needed in a band of grid points around the surface.

4 RESULTS

In Fig. 2-a we show an example of a 3D synthetic surface given by the implicit formula

$$F(x,y,z) = 1 - |L - \max(|x-L|, |y-L|, |z-L|)| + \qquad (10)$$
$$2(\sin Kx)(\sin Ky) = 0.$$

The first two terms produce a hollow cubic box, and the last term introduces a modulation on the shape of the surface. In this example, $L = 15$ and $K = \pi/3$ and the discretization has size $50 \times 50 \times 50$ grid points. We painted the surface with RGB components proportional to the unit surface normal, in order to reveal the texture of the shape. Note the holes going through the shape. The sphere on the bottom shows the directions of the normals on the surface. In Fig. 2-b we corrupted the implicit function by adding a random number of uniform deviate in the interval $(-3, 3)$ to each grid-point. In Fig. 2-c we denoised the surface with a Wiener filter with $N_0 = 100$ and a window of size $M = 16$. (The computation time is about 4 seconds on a 550 MHz PC.) In order to put this result in context with other elementary methods, we compare the results with Gaussian and curvature flow denoising. In Fig. 3-a we denoised the same image with a Gaussian filter, implemented by running a heat equation $F_t = \Delta F$ during the time $t = 0.6$ (this is equivalent to convolution with a Gaussian kernel). This filter does not preserve the shape texture very well. Note the lack of periodicity on the holes on the endcap of the box, and the spurious bridges between the vertical columns on the walls. In Fig. 3-b, we smoothed F with a curvature flow given by $F_t = H|\nabla F|$ [12] during the time $t = 0.75$, where H is the mean curvature of the level surfaces $F = const$. The local character of the method implies that it cannot "see" the texture of the surface. This simple example illustrates the importance of having a method that preserves both the shape texture and periodicity.

Next in Fig. 4-a, we show a triangulated surface, which was extracted with the marching cubes algorithm from the implicit surface of of the previous figure. It has 97000 oriented triangles which separate the space in complementary interior and an exterior regions. We computed the distance function with the hybrid fast marching method described above. We then smoothed it with the Wiener filter with $N_0 = 2$. In Fig. 4-b, we show the resulting smoothed surface. In Fig. 5, we show a slice of the distance function before and after denoising. Note that the noise on the distance function is concentrated around the surface, while farther from the surface, the solution of the Eikonal equation is smoother (essentially because the formation of multiple shocks tends to smooth out ϕ).

All the following examples use as input public-domain triangulated surfaces. We first compute the implicit representation of the given surface, and then process it.

In Fig. 6-a, we show a surface representing a human hand skeleton, obtained from the original of the Georgia Institute of Technology Large Geometric Models Archive. In Fig. 6-b, we produced a corrupted triangulated surface by adding random numbers of uniform deviate in $(-2, 2)$ to the distance function of Fig.

Figure 1: *Illustration of the texture preservation property of the Wiener filter. The original image (top) is corrupted adding noise to each RGB component (center) and then is restored applying the Wiener filter to the each RGB component.*

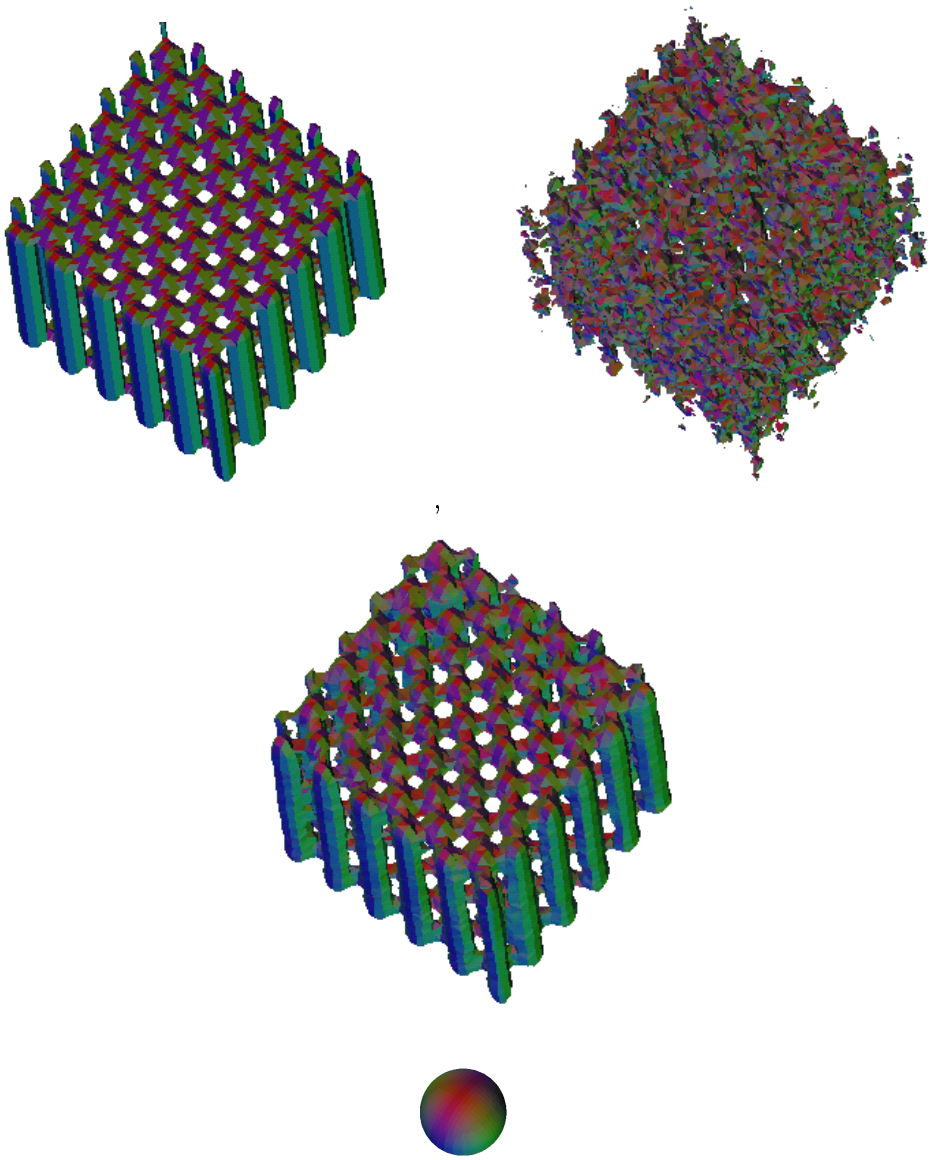


Figure 2: *a) Example of an implicit surface with texture on the shape (top left). b) With added noise to the implicit function (top right). c) Restored with the Wiener filter (bottom). The sphere is to show the directions of the normals.*

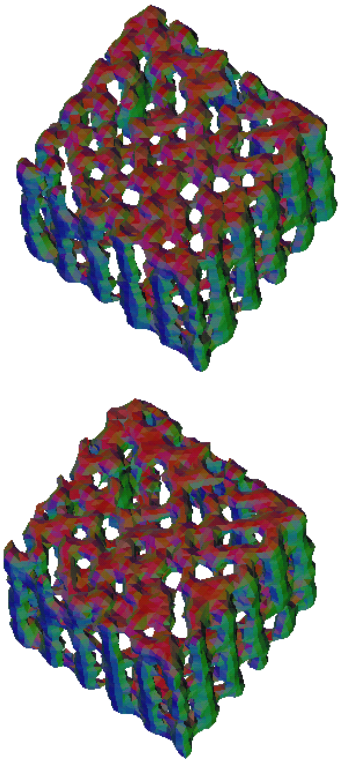


Figure 3: *a) Same noisy surface smoothed with a Gaussian filter (top) and b) with a curvature flow (bottom).*

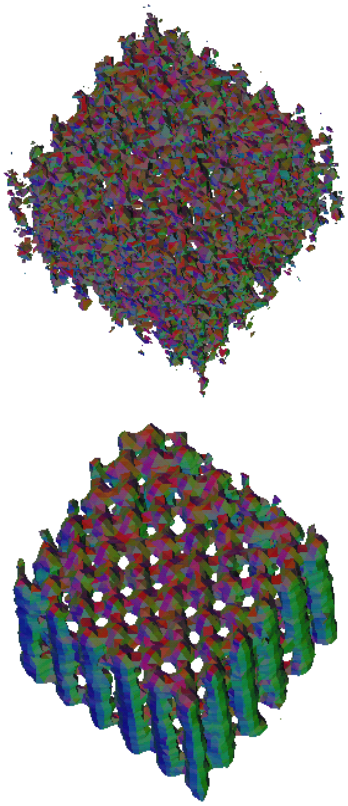


Figure 4: *a) Triangulated noisy surface smoothed by first integrating the Eikonal equation and then applying the Wiener filter (b).*

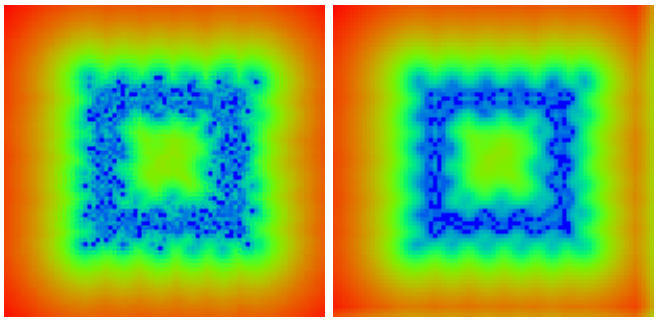


Figure 5: *a) Section of the distance function computed with the noisy surface of the previous figure. b) Implicit function resulting after denoising.*

6-a, and then extracting a triangulation with marching cubes. Note that this noise simulation introduces noise in the original topology of the surface. In Fig. 6-c, we smooth out the surface by computing the distance function to the corrupted triangulation and then applying a curvature flow ($t = 0.6$). In Fig. 6-d, we restored it by using the Wiener filter, with $M = 8, N_0 = 10$. The size of the discretization of the domain is $165 \times 125 \times 77$. We note that both results are very similar. This indicates that the approach proposed here enjoys the added qualities of simplicity and shape periodicity/texture preservation without sacrificing important attributes obtained by curvature-driven flow smoothing techniques.

In Fig. 7-a, we have the orthographic projection of another triangulated surface. This particular shape is interesting because of the scales and the spines. We constructed a corrupted triangulated surface by first computing the distance function, then adding random numbers of uniform deviate in $(-0.5, 0.5)$ to the distance function, and then extracting the triangulation with marching cubes; see Fig. 7-b. In Fig. 7-c, we show the smoothed surface after denoising with $N_0 = 0.5$ and $M = 16$. The grid size is $258 \times 190 \times 133$.

5 CONCLUSIONS

In this paper, we introduced a simple algorithm that removes noise from triangulated and implicit surfaces with local periodicity. A key technical problem that we had to solve was how to apply the Wiener filter with the fast Fourier transform to surfaces of arbitrary shape.

Our proposed method uses more information than the local derivatives of PDE based methods, in the sense that it employs a collection of M points (the window size) in the neighborhood of every point of the surface. In this way the information about the local modes of the texture is utilized in the computation.

To conclude, we now summarize the limitations of our present implementation and different ways to generalize it:

1) The topology of S may be arbitrary in principle, however in practice the best results are achieved if S separates the Euclidean space into an exterior and an interior region. That means that a continuous signed distance function may be computed, with $\phi < 0$ inside the surface and $\phi > 0$ outside.

2) There is no fundamental reason to restrict the computation to the simplifications that we presented in Eqs. 8 which we did for illustrative purposes. If the statistics of the noise are known for a particular field of application, its structure may be explicitly included in the Wiener filter, as well as the correlation between noise and signal [8].

3) As previously mentioned, any other method may be used to compute the distance function to the initial surface. For example, the surface could be approximated by a cloud of points instead of triangles [25].

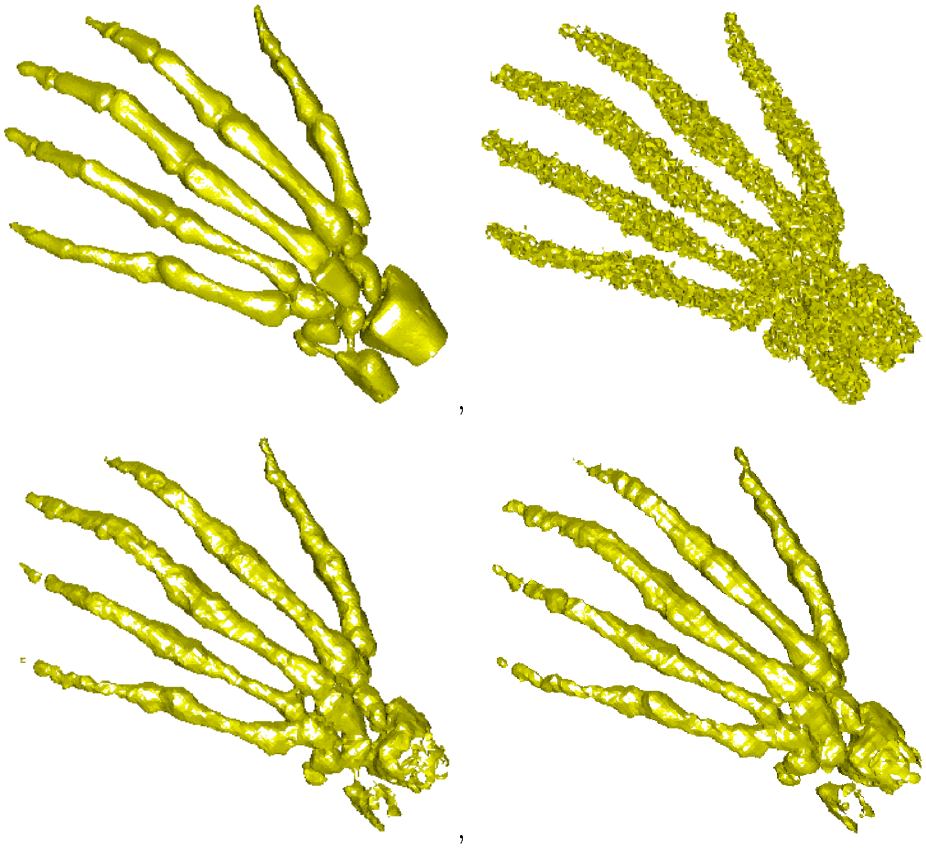


Figure 6: *a) Original triangulated surface, b) with added noise, c) restored with curvature flow and d) restored with the Wiener filter (Lexicographic order).*

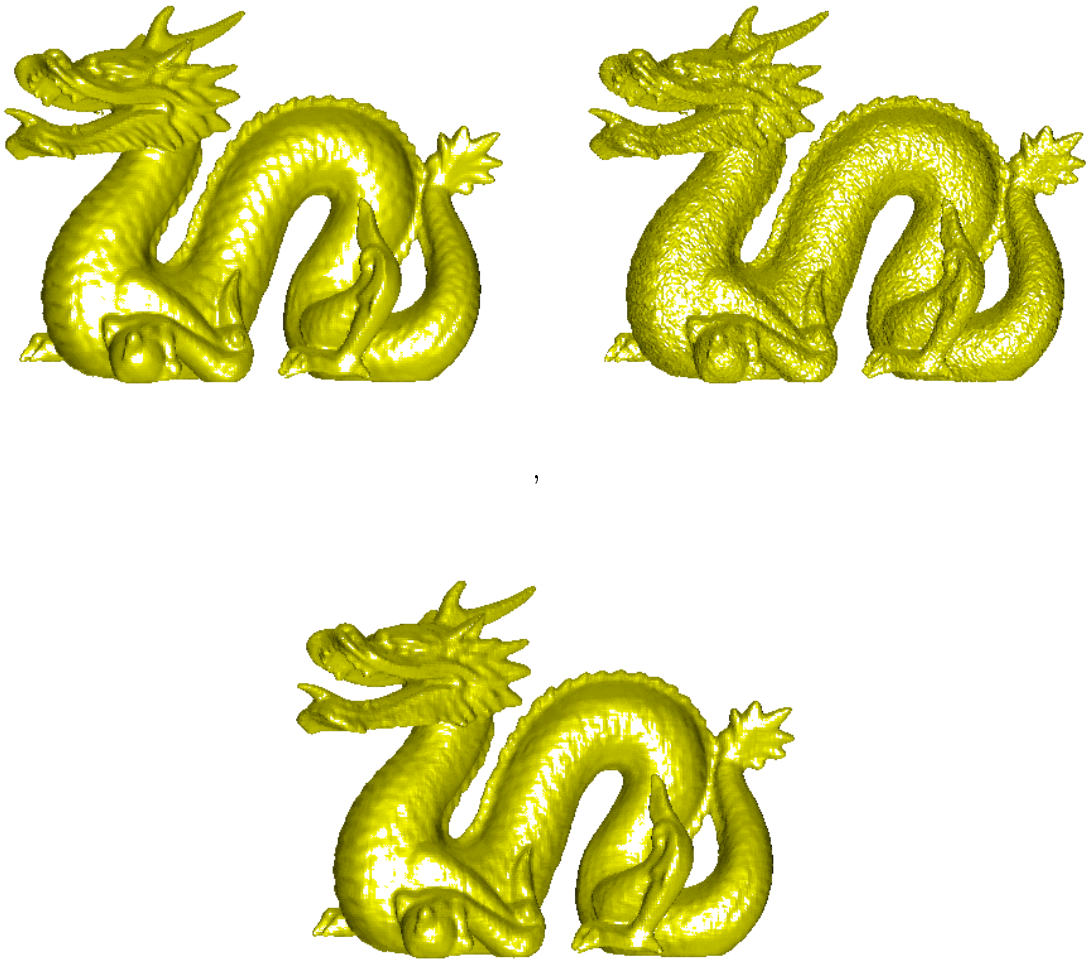


Figure 7: *a) Original triangulated surface, b) with added noise and c) restored.*

References

[1] U. Clarenz, U. Diewald, M. Rumpf, "Anisotropic diffusion in surface processing," *Proceedings IEEE Visualization 2000*, 2000.

[2] M. Desbrun, M. Meyer, P. Schroeder P, and A. Barr, "Discrete differential-geometry operators in nD," from http://www.multires.caltech.edu/pubs/, 2000.

[3] M. Desbrun, M. Meyer, P. Schroeder, and A. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," *SIGGRAPH'99*, Los Angeles, California, pp. 46–52, 1999.

[4] M. Eck and H. Hoppe, "Automatic reconstruction of B-spline surfaces of arbitrary topological type," *Computer Graphics*, 1996.

[5] I. Guskov, W. Sweldens, and P. Schroeder, "Multiresolution Signal Processing for Meshes", *SIGGRAPH'99*, Los Angeles, California, 1999.

[6] G. Granlund and H. Knuttson, *Signal Processing for Computer Vision*, Kluwer Academic Publishers, Boston, 1995.

[7] J. Helmsen, E. G. Puckett, P. Collela, and M. Dorr, "Two new methods for simulating photolithography development in 3D," *Proc. SPIE Microlithography* **IX**, pp. 253, 1996.

[8] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, New York, 1989.

[9] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," *Computer Graphics*, pp. 313-324, 1996.

[10] S. Mauch, "Closest point transform," from http://www.ama.caltech.edu/~seanm/software/cpt/cpt.html, 2000.

[11] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Prentice Hall, New York, 1996.

[12] S. Osher and J. Sethian, "Fronts propagating with curvature dependent speed: algorithms based on Hamilton-Jacobi formulation," *Journal of Computational Physics* **79**, pp. 12-49, 1988.

[13] A. Papoulis, *Probability, Random Variables and Stochastic Processes*, McGraw Hill, New York, 1965.

[14] D. Peng, B. Merriman, S. Osher, H. Zhao, M. Kang, "A PDE-based fast local level set method," *Journal of Computational Physics* **155**, pp. 410-438, 1999.

[15] T. Preusser and M. Rumpf, "A level set method for anisotropic geometric diffusion in 3D image processing," December 2000, pre-print (courtesy of the authors).

[16] J. Sethian, "Fast marching level set methods for three-dimensional photolithography development," *Proc. SPIE International Symposium on Microlithography*, Santa Clara, California, March, 1996.

[17] J. A. Sethian, "A fast marching level-set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci.* **93:4**, pp. 1591-1595, 1996.

[18] J. Sethian, *Level Set Methods and Fast Marching Methods*, Cambridge University Press, Cambridge Monographs on Applied and Computational Mathematics, Cambridge, UK, 1999.

[19] G. Taubin, "Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation," *IEEE Trans. PAMI* **13:11**, pp. 1115-1138, 1991.

[20] G. Taubin, "A signal processing approach to fair surface design," *SIGGRAPH'95*, 1995.

[21] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control* **40**, pp. 1528-1538, 1995.

[22] C. F. Westin, J. Richolt, V. Moharir, and R. Kikinis, "Affine adaptive filtering of CT data," *Medical Image Analysis* **4**, pp. 161-177, 2000.

[23] R. Tsai, "Rapid and accurate computation of the distance function using grids", from http://www.math.ucla.edu/∼ytsai/math-page/index.shtml, 2000.

[24] G. Yngve and G. Turk, "Creating smooth implicit surfaces from polygonal meshes," Technical Report GIT-GVU-99-42, Graphics, Visualization, and Usability Center, Georgia Institute of Technology, from http://www.cc.gatech.edu/gvu/geometry/publications.html, 1999.

[25] H. Zhao, S. Osher, B. Merriman, and M. Kang, "Implicit, non-parametric shape reconstruction from unorganized points using a variational level set method," *Comp. Vision and Image Understanding* **80**, pp. 295-314, 2000.

Solving Variational Problems and Partial Differential Equations Mapping into General Target Manifolds

Facundo Mémoli [*] **Guillermo Sapiro**[†] **Stanley Osher**[‡]

Abstract

A framework for solving variational problems and partial differential equations that define maps onto a given generic manifold is introduced in this paper. We discuss the framework for arbitrary target manifolds, while the domain manifold problem was addressed in [3]. The key idea is to implicitly represent the target manifold as the level-set of a higher dimensional function, and then implement the equations in the Cartesian coordinate system of this new embedding function. In the case of variational problem, we restrict the search of the minimizing map to the class of maps whose target is the level-set of interest. In the case of partial differential equations, we implicitly represent all the equation characteristics. We then obtain a set of equations that while defined on the whole Euclidean space, they are intrinsic to the implicit target manifold and map into it. This permits the use of classical numerical techniques in Cartesian grids, regardless of the geometry of the target manifold. The extension to open surfaces and submanifolds is addressed in this paper as well. In the latter case, the submanifold is defined as the intersection of two higher dimensional surfaces, and all the computations are restricted to this intersection. Examples of the applications of the framework here described include harmonic maps in liquid crystals, where the target manifold is an hypersphere; probability maps, where the target manifold is an hyperplane; chroma enhancement; texture mapping; and general geometric mapping between high dimensional surfaces.

[*]Instituto de Ingenieria Electrica, Universidad de la Republica, Montevideo, Uruguay.

[†]Corresponding author: Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, guille@ece.umn.edu

[‡]UCLA Mathematics Department, Los Angeles, CA 90095.

1 Introduction

In a number of applications in mathematical physics, image processing, computer graphics, and medical imaging, we have to solve variational problems and partial differential equations defined on a general manifold $\mathcal{M}$ (*domain manifold*), mapping the data onto another general manifold $\mathcal{N}$ (*target manifold*). That is, we deal with maps from $\mathcal{M}$ to $\mathcal{N}$. When these manifolds are for example three dimensional surfaces, the implementation of the corresponding gradient descent flow or the given PDE's is considerably elaborated. In [3] we have shown how to address this problem for general domain manifolds, while restricting the target manifolds $\mathcal{N}$ to the trivial cases of the Euclidean space or hyper-spheres. The key idea was to implicitly represent the domain surface as the (zero) level-set of a higher dimensional function ϕ, and solve the PDE in the Cartesian coordinate system of this new embedding function. The technique was justified and demonstrated in [3]. It is the goal of this paper to show how to work with general target manifolds, and not just hyper-planes or hyper-spheres as previously reported in the literature. Inspired by [3], we also embed the target manifold $\mathcal{N}$ as the (zero) level-set of a higher dimensional function ψ. That is, when solving the gradient descent flow (or in general, the PDE), we guarantee that the map receives its values on the zero level-set of ψ. The map is defined on the whole space, although it never receives values outside of this level-set. Examples of applications of this framework include harmonic maps in liquid crystals ($\mathcal{N}$ is an hypersphere) and 3D surface warping [43]. In this last case, the basic idea is to find a smooth map between two given three dimensional surfaces. Due to the lack of the new frameworks introduced here and in [3], this problem is generally addressed in the literature after an intermediate mapping of the surfaces onto the plane is performed (see also [25, 46]). With these novel frameworks, direct three dimensional maps can be computed without any intermediate mapping, thereby eliminating their corresponding geometric distortions [31]. For this application, as in [43], boundary conditions are needed, and how to add them to the frameworks introduced here and in [3] is addressed in [31].

To introduce the ideas, in this paper we concentrate on flat domain manifolds.[1] When combining this framework with the results on [3], we can of course work with general domains and then completely avoid other popular surface representations, like triangulated surfaces. We are then able to work with intrinsic equations, in Euclidean space and with classical numerics on Cartesian grids, regardless of the geometry of the involved domain and target manifolds. In addition to presenting the general theory, we also address the problem of target submanifolds and open surfaces. A number of theoretical results complement the algorithmic framework here described.

The implicit representation of surfaces here introduced for solving variational problems and PDE's is inspired in part by the level-set work of Osher and Sethian [33]. This work, and those that followed it, showed the importance of embedding deforming surfaces in higher dimensional functions, obtaining more robust and accurate numerical algorithms (and topological freedom). Note that in contrast with the level-set approach of Osher and Sethian, our target manifold is fixed, what is "deforming" is the dataset being mapped onto it.

Numerical schemes that solve gradient descent flows and PDE's onto generic target manifolds $\mathcal{N}$ (and spheres or surfaces in particular) will in general move the points outside of $\mathcal{N}$ due to numerical errors. The points will then need to be projected back,[2] see for example [1, 9] for the case of $\mathcal{N}$ being a sphere (where the projection is trivial, just a normalization). For general target manifolds, this projection means that for every point $p \in \mathbb{R}^d$ ($\mathcal{N} \subset \mathbb{R}^d$) we need to know the closest point to p in $\mathcal{N}$. This means knowing the

[1] For completeness, we will present the general equations for both generic domain and target manifolds at the end of the paper. These equations are easily derived from [3] and the work presented in this paper.

[2] For particular flat target manifolds as the whole space $\mathbb{R}^d$ or as those in [34], the projection is not needed. Other authors, e.g., [6, 26], have avoided the projection step for particular cases, while in [48] the authors modify the given variational formulation to include the projection step.

distance from every point $p \in I\!R^d$ to $\mathcal{N}$ (or at least all points in a band of $\mathcal{N}$). This is nothing else than an implicit representation of the target $\mathcal{N}$, being the particular embedding a distance function. This presents an additional justification for the framework here introduced.

In a number of applications, the surfaces are already given in implicit form, e.g., [5], therefore, the framework introduced in this paper it is not only simple and robust, but it is also natural in those applications. On the other hand, not all surfaces (manifolds) are originally represented in implicit form. When the target manifold $\mathcal{N}$ is simple, like hyper-spheres in the case of liquid crystals, the implicitation process is trivial. For generic surfaces, we need to apply an algorithm that transforms the given explicit representation into an implicit one. Although this is still a very active area of research, many very good algorithms have been developed, e.g., [14, 18, 27, 45].

2 The Framework

From now we assume that the target manifold $\mathcal{N}$ is given as the zero level set of a higher dimensional embedding $\psi : I\!R^d \to I\!R$, which we consider to be a distance function (this mainly simplifies the notation). For the case where $\mathcal{N}$ is a surface in three dimensional space for example, then $\psi : I\!R^3 \to I\!R$. We also assume that the domain manifold $\mathcal{M}$ is flat and open (as mentioned in the introduction, general domain manifolds were addressed in [3]). We illustrate the basic ideas with a functional from the theory of harmonic maps. This is just a particular example (and a very important one), and from it will be clear how the same arguments can be applied to any given variational problem and PDE. In particular, it can be applied to common Navier-Stokes flows used in brain warping [31].

2.1 Variational Formulation

We search for necessary conditions for the functional $E[\vec{u}]$, defined by

$$E[\vec{u}] \triangleq \int_{\mathcal{M}} e[\vec{u}]\, d_{\mathcal{M}} v \tag{1}$$

where

$$e[\vec{u}] \triangleq \frac{1}{2} \|\mathbf{J}_{\vec{u}}\|^2_{\mathcal{F}} \tag{2}$$

to achieve a minimum. Here, $\|\cdot\|^2_{\mathcal{F}} = \sum_{ij} (\cdot)^2_{ij}$ is the norm of Frobenius and $\mathbf{J}_{\vec{u}}$ is the Jacobian of the map $\vec{u} : \mathcal{M} \to \{\psi = 0\}$. Note that here we are already restricting the map to be onto the zero level-set of ψ, that is, onto the surface of interest $\mathcal{N}$ (the target manifold). This is what permits us to work with the embedding function and the whole space, while guaranteeing that the map will always be onto the target manifold, as desired. We use $\vec{\cdot}$ to note that for the most general case, the function is vectorial. Once again, this energy will be used throughout this paper to exemplify our framework. It will be clear after developing this example that the same arguments work for other variational formulations, as well as for generic PDE's defined onto generic surfaces.

Assume that $\vec{u}$ is a map minimizing $E(\cdot)$. Given $t > 0$, we construct the variation

$$\vec{v}_t \triangleq \vec{u} + t\, \vec{r}$$

where $\vec{r}$ is a compact map C^∞ in $\mathcal{M}$. For an arbitrary $x \in \mathcal{M}$, we will in general not obtain that $\vec{v}_t(x) \in \{\psi = 0\}$, that is, $\psi(\vec{v}_t(x)) \neq 0$. Therefore, this variation is not admissible. On the other hand, we can from it construct an admissible variation via

$$\vec{w}_t \stackrel{\triangle}{=} \mathbf{\Pi}_{\{\psi=0\}}(\vec{v}_t)$$

where $\mathbf{\Pi}_{\{\psi=0\}} : R^d \rightarrow \{\psi = 0\}$ is the *projection operator* onto $\{\psi = 0\}$. Note that since ψ is a signed distance function, we can simply write this projection operator onto $\{\psi = 0\}$ as

$$\mathbf{\Pi}_{\{\psi=0\}}(\vec{\alpha}) = \vec{\alpha} - \psi(\vec{\alpha})\ \nabla\psi(\vec{\alpha}).$$

Let's now define

$$\mathcal{E}(t) \stackrel{\triangle}{=} E[\vec{w}_t]$$

Since the energy achieves a minimum for $t = 0$,

$$\dot{\mathcal{E}}_0 \stackrel{\triangle}{=} \left.\frac{dE(t)}{dt}\right|_{(t=0)} = 0.$$

Let's compute this first variation. We have that

$$\dot{\mathcal{E}}_0 = \sum_{ij} \int_{\mathcal{M}} \left(\frac{\partial w_t^i}{\partial x_j} \frac{d\left(\frac{\partial w_t^i}{\partial x_j}\right)}{dt} \right)\Bigg|_{t=0} d_{\mathcal{M}} v \tag{3}$$

Moreover ($\mathbf{H}_\psi$ stands for the Hessian of ψ),

$$\begin{aligned} \frac{\partial w_t}{\partial x_j} &= \left(\frac{\partial \vec{u}}{\partial x_j} + t\frac{\partial \vec{r}}{\partial x_j}\right) - \left(\nabla\psi(\vec{w}_t) \cdot \left(\frac{\partial \vec{u}}{\partial x_j} + t\frac{\partial \vec{r}}{\partial x_j}\right)\right) \nabla\psi(\vec{w}_t) \\ &- \psi(\vec{w}_t)\, \mathbf{H}_\psi(\vec{w}_t) \left(\frac{\partial \vec{u}}{\partial x_j} + t\frac{\partial \vec{r}}{\partial x_j}\right) \end{aligned} \tag{4}$$

and we observe that

$$\left.\frac{\partial w_t}{\partial x_j}\right|_{(t=0)} = \frac{\partial \vec{u}}{\partial x_j} - \left(\nabla\psi(\vec{u}) \cdot \frac{\partial \vec{u}}{\partial x_j}\right) \nabla\psi(\vec{u})$$

since $\psi(\vec{u}) = 0$. We can further simplify this observing that $0 = \frac{\partial \psi(\vec{u})}{\partial x_j} = \nabla\psi(\vec{u}) \cdot \frac{\partial \vec{u}}{\partial x_j}$. Therefore,

$$\left.\frac{\partial w_t}{\partial x_j}\right|_{(t=0)} = \frac{\partial \vec{u}}{\partial x_j} \tag{5}$$

With a bit of further simple analysis we can compute the additional derivative, $\frac{d\left(\frac{\partial w_t^i}{\partial x_j}\right)}{dt} = \frac{\partial\left(\frac{dw_t^i}{dt}\right)}{\partial x_j}$. This change in the order of derivatives is done in order to immediately evaluate the result at $t = 0$, thereby simplifying the following derivative. Following in an similar form, we obtain

$$\frac{dw_t^i}{dt} = \vec{r} - (\nabla\psi(\vec{w}_t) \cdot \vec{r})\, \nabla\psi(\vec{w}_t) - \psi(\vec{w}_t)\ \mathbf{H}_\psi(\vec{w}_t)\vec{r} \tag{6}$$

and

$$\left.\frac{dw_t^i}{dt}\right|_{(t=0)} = \vec{r} - (\nabla\psi(\vec{u}) \cdot \vec{r})\, \nabla\psi(\vec{u}). \tag{7}$$

Combining the above computations all together we obtain

$$\begin{aligned}\left.\frac{d\left(\frac{\partial w_t^i}{\partial x_j}\right)}{dt}\right|_{(t=0)} &= \frac{\partial\left(\left.\frac{dw_t^i}{dt}\right|_{(t=0)}\right)}{\partial x_j} \\ &= \frac{\partial\vec{r}}{\partial x_j} - \nabla\psi(\vec{u})\left\{\frac{\partial\vec{r}}{\partial x_j}\cdot\nabla\psi(\vec{u}) + \mathbf{H}_\psi\left(\vec{r},\frac{\partial\vec{u}}{\partial x_j}\right)\right\} - \left(\vec{r}\cdot\nabla\psi(\vec{u})\right)\left(\mathbf{H}_\psi\,\frac{\partial\vec{u}}{\partial x_j}\right).\end{aligned} \tag{8}$$

Following from (3) we have that[3]

$$\begin{aligned}\dot{\mathcal{E}}_0 &= \sum_j\int_{\mathcal{M}}\left(\frac{\partial\vec{w}_t}{\partial x_j}\frac{d\left(\frac{\partial\vec{w}_t}{\partial x_j}\right)}{dt}\right)\Bigg|_{t=0} d_{\mathcal{M}}v \\ &= \sum_j\int_{\mathcal{M}}\left\{\frac{\partial\vec{r}}{\partial x_j}\cdot\frac{\partial\vec{u}}{\partial x_j} - \left(\vec{r}\cdot\nabla\psi(\vec{u})\right)\mathbf{H}_\psi\left[\frac{\partial\vec{u}}{\partial x_j},\frac{\partial\vec{u}}{\partial x_j}\right]\right\} d_{\mathcal{M}}v.\end{aligned} \tag{9}$$

Now, applying the divergence theorem we conclude the computation. We first write

$$\sum_{ij}\int_{\mathcal{M}}\frac{\partial\vec{r}}{\partial x_j}\cdot\frac{\partial\vec{u}}{\partial x_j}\;d_{\mathcal{M}}v = \sum_i\int_{\mathcal{M}}\nabla r^i\cdot\nabla u^i\;d_{\mathcal{M}}v$$

and then apply the fact $\nabla r^i\cdot\nabla u^i = \nabla\cdot\left(r^i\nabla u^i\right) - r^i\,\Delta u^i$, together with the divergence theorem, to obtain ($\mathbf{n}$ stands for the outward unit normal to $\partial\mathcal{M}$).

$$\sum_{ij}\int_{\mathcal{M}}\frac{\partial\vec{r}}{\partial x_j}\cdot\frac{\partial\vec{u}}{\partial x_j}\;d_{\mathcal{M}}v = \sum_i\int_{\partial\mathcal{M}} r^i\frac{\partial u^i}{\partial\mathbf{n}}dS - \int_{\mathcal{M}} r^i\Delta u^i d_{\mathcal{M}}v \tag{10}$$

To conclude we put together this last expression with (8), and after some algebra we obtain that $\dot{\mathcal{E}}_0$ is equal to

$$\int_{\partial\mathcal{M}}\vec{r}\cdot\mathbf{J}_{\vec{u}}\mathbf{n}\;dS - \int_{\mathcal{M}}\vec{r}\cdot\left\{\Delta\vec{u} + \left(\sum_k\mathbf{H}_\psi\left[\frac{\partial\vec{u}}{\partial x_k},\frac{\partial\vec{u}}{\partial x_k}\right]\right)\nabla\psi(\vec{u})\right\} d_{\mathcal{M}}v \tag{11}$$

The boundary condition is eliminated since the support of $\vec{r}$ is compactly included in $\mathcal{M}$. To eliminate the additional term for an arbitrary $\vec{r}$ we must impose

$$\Delta\vec{u} + \left(\sum_k\mathbf{H}_\psi\left[\frac{\partial\vec{u}}{\partial x_k},\frac{\partial\vec{u}}{\partial x_k}\right]\right)\nabla\psi(\vec{u}) = 0. \tag{12}$$

This gives the corresponding Euler-Lagrange for the given variational problem. Note once again from our computations that in spite that all the terms "live" in the Euclidean space embedding the target manifold, $\vec{u}$ will always map onto the level-set of interest, $\{\psi = 0\}$, and therefore, onto the surface of interest. This is guaranteed by this equation, no additional computations are needed. This is the beauty of the approach, while working freely on the Euclidean space (and therefore with Cartesian numerics), we can guarantee that the equations are intrinsic to the given surfaces of interest. We will further verify this in §2.4 to help the reader with the intuition behind this framework.

[3]We have used as before the notation $A\left[\vec{x},\vec{y}\right] = \vec{y}^T A\,\vec{x}$

2.2 Harmonic Maps

The expressions derived in the previous sections come from the theory of harmonic maps, e.g., [4, 7, 11, 13, 15, 16, 20, 23, 35, 38, 39, 40]. In general, harmonic maps are defined as maps between two manifolds $(\mathcal{M}, g)$ and $(\mathcal{N}, h)$ minimizing the energy

$$E[\vec{u}] \triangleq \int_{\mathcal{M}} e[\vec{u}]\ dV_{\mathcal{M}} \tag{13}$$

where in local coordinates the *energy density* $e[\vec{u}]$ is given by

$$e[\vec{u}](x) \triangleq \frac{1}{2}\ g^{pq}(x)\ h_{ij}(\vec{u}(x))\ \frac{\partial u^i}{\partial x_p}\ \frac{\partial u^j}{\partial x_q}. \tag{14}$$

We have used Einstein's summation here, where repeated indices indicate summation with respect to this index, together with the usual notation for tensors.[4] When both the domain and target manifolds are represented explicitly, the classical case, the Euler-Lagrange equation corresponding to this energy is given by (see [38])

$$\Delta_{\mathcal{M}} u^l + \mathbf{\Gamma}^l_{ij}(\vec{u})\ g^{\alpha\beta}\frac{\partial u^i}{\partial x^\alpha}\frac{\partial u^j}{\partial x^\beta} = 0 \tag{15}$$

where $\Delta_{\mathcal{M}}$ is the Laplace-Beltrami operator (reduced to the regular Laplacian for the case of flat domain manifolds) and $\mathbf{\Gamma}^l_{ij}(\vec{u})$ stands for the Christoffel symbols of the target manifold evaluated at $\vec{u}$. Note that the first component, the Laplace-Beltrami, addresses the domain manifold, while the second term addresses the target manifold. By embedding the target manifold, we are changing the Christoffel symbols (expressing them in implicit form, see below),[5] while the work in [3] changed the other terms, since the embedding was done to the domain manifold, see §5.

As an example, let's see what happens with the above energy for the Euclidean case. Since both metrics are proportional to the identity,

$$e[\vec{u}](x) = \frac{K}{2}\sum_{ij}\left(\frac{\partial u^i}{\partial x_j}\right)^2$$

which is just a constant multiplying $\|\mathbf{J}_{\vec{u}}\|^2_{\mathcal{F}}$. Therefore, the energy defined in the previous case is just a particular case of harmonic maps. In general, this energy can be used in problems such as color image denoising and directions denoising [40, 41], as a regularization term for ill-possed problems defined on general surfaces [17], for general denoising [37, 44], for models of liquid crystals, and as a component of a system for surface mapping and matching [13, 31, 46].

2.2.1 An(other) Informal Calculation

We now present an additional computation that connects in a deep way the implicit framework with harmonic maps. We consider the harmonic energy density given in (14) for the planar domain manifold case ($g_{ij} = \delta_{ij}$). We can simplify things to obtain

$$e[\vec{u}](x) = \frac{1}{2}\ h_{ij}(\vec{u}(x))\ \frac{\partial u^i}{\partial x_p}\ \frac{\partial u^j}{\partial x_p} = \frac{1}{2}\sum_p \mathbf{h}[\vec{u}_{x_p}, \vec{u}_{x_p}]$$

[4] $(g^{-1})_{ij} \triangleq g^{ij}$.

[5] Or alternatively, the second fundamental form of the target manifold.

We know that $\mathbf{\Pi}_{\nabla\psi} = \mathbf{I} - \nabla\psi\nabla\psi^T$ can be thought of as the inverse of the target manifold's *metric tensor*. But since $\nabla\psi$ is a zero eigenvalue eigenvector for $\mathbf{\Pi}_{\nabla\psi}$, it will be a ∞ eigenvalue eigenvector for $\mathbf{\Pi}_{\nabla\psi}^{-1}$. Then, we can't use the identification $\mathbf{h} = (h_{ij}) \leftrightarrow \mathbf{\Pi}_{\nabla\psi}^{-1}$ in the above expression for the energy density. However, we can proceed as follows. Take $\epsilon > 1$ and define the metric[6]

$$\mathbf{h}^\epsilon \stackrel{\triangle}{=} \left(\epsilon\mathbf{I} - \nabla\psi\nabla\psi^T\right)^{-1}$$

one can then compute the inverse as (it's an elementary formula, see for example [24])

$$\mathbf{h}^\epsilon = \frac{1}{\epsilon}\left(\mathbf{I} + \frac{\nabla\psi\nabla\psi^T}{\epsilon - 1}\right)$$

The energy density can be rewritten as (we will use a subindex ϵ)

$$e_\epsilon[\vec{u}](x) = \frac{1}{2\epsilon}\left(\sum_i \|\vec{u}_{x_i}\|^2 + \frac{1}{\epsilon-1}\sum_i |\vec{u}_{x_i}\cdot\nabla\psi|^2\right)$$

After computing the variational derivative for the functional $\int_{\mathcal{M}} e_\epsilon[\vec{u}](x)\,dx$ we obtain that $\vec{u}$ must satisfy

$$\Delta\vec{u} + \frac{1}{\epsilon-1}\left(\sum_i \mathbf{H}_\psi[\vec{u}_{x_i}, \vec{u}_{x_i}] + \Delta\vec{u}\cdot\nabla\psi\right)\nabla\psi = 0$$

By multiplying all the terms in the above equation by $\epsilon - 1$ and letting $\epsilon \to 1$ we find that the expression between brackets must vanish. As we will see in §2.4, what's between brackets is nothing but $\Delta\nu(x)$ where $\nu(x) = \psi(\vec{u}(x))$. So ν is a harmonic function in $\mathcal{M}$. It is also evident that ν satisfies Dirichlet boundary conditions if $\vec{u}$ does, and since we are trying to map things from $\mathcal{M}$ to $\mathcal{N}$, those boundary conditions for $\vec{u}$ must be such that $dist(\vec{u}(x), \mathcal{N}) = 0$ for $x \in \mathcal{M}$, so $\nu|_{\partial\mathcal{M}} = 0$. Then we conclude that ν must be zero everywhere in $\mathcal{M}$.

2.3 The Mapping Flow

The PDE used for solving the harmonic energy is given by its corresponding *gradient descent* flow. This gradient descent is given by

$$\frac{\partial u^i}{\partial t} = \Delta u^i + \sum_{k=1}^{d} \mathbf{H}_\psi(\vec{u})\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]\frac{\partial\psi}{\partial u^i}(\vec{u}) \tag{16}$$

where the initial datum $\vec{u}_0$ is given by the vector field we want to process, together with *Neumann* boundary conditions:

$$\begin{cases} \vec{u}(x,0) = \vec{u}_0(x), \;\; x \in \mathcal{M} \\ \mathbf{J}_{\vec{u}}\mathbf{n}|_{\partial\mathcal{M}} = \mathbf{0}. \end{cases} \tag{17}$$

The use of *Neumann* boundary conditions needs to be justified. In the scalar case, one has the evolution problem

$$\begin{cases} I_t(x,t) = \Delta I(x,t) \;\; x \in \mathcal{M},\; t \geq 0 \\ I(x,0) = I_0(x), \;\; x \in \mathcal{M} \\ \nabla I \cdot \mathbf{n}|_{\partial\mathcal{M}} = \mathbf{0}. \end{cases} \tag{18}$$

[6] Since $\epsilon > 1$ all the eigenvalues are positive.

We observe that the quantity $\sigma(t) \stackrel{\triangle}{=} \int_{\mathcal{M}} I(x,t)\, d_{\mathcal{M}} v$ remains constant,

$$\dot{\sigma}(t) = \int_{\mathcal{M}} I_t(x,t)\, d_{\mathcal{M}} v = \int_{\mathcal{M}} \Delta I(x,t)\, d_{\mathcal{M}} v = \int_{\mathcal{M}} \nabla \cdot (\nabla I)\, d_{\mathcal{M}} v = \int_{\partial \mathcal{M}} \nabla I \cdot \mathbf{n}\; d_{\mathcal{M}} s = 0$$

thereby imposing the boundary conditions.

One wonders which quantity is preserved thru time by the flow in the general case, when imposing the boundary condition (17). We illustrate this for the particular case of $\mathcal{N} = S^1$. In this case, the evolution equations are given by (see also §2.4 below)

$$\begin{cases} X_t = \Delta X + (\|\nabla X\|^2 + \|\nabla Y\|^2) X \\ Y_t = \Delta Y + (\|\nabla X\|^2 + \|\nabla Y\|^2) Y \end{cases} \tag{19}$$

The Neumann boundary conditions for this case are written as

$$\nabla X \cdot \mathbf{n} = \nabla Y \cdot \mathbf{n} = 0 \;\text{ in } \partial \mathcal{M}$$

Transforming to polar coordinates (ρ, θ) one finds that the evolution equations (for smooth initial data, and at least for some time) are (see also [35])

$$\begin{cases} \theta_t = \Delta \theta \\ \rho_t = 0 \end{cases} \tag{20}$$

with boundary conditions

$$\nabla \theta \cdot \mathbf{n} = 0 \;\text{ in } \partial \mathcal{M}$$

Again one finds that $\int_{\mathcal{M}} \theta(x,t)\, d_{\mathcal{M}} v$ is *constant.*

In the most general case, when the target manifold is arbitrary, one might guess that the *intrinsic barycenter*[7] of the map is preserved through time, since that's exactly what the particular cases given above show us. However, to the best of our knowledge, there is not such a result in the literature of harmonic maps, and the conservation of the barycenter is only obtained when constraints are added. The examples discussed above still motivate the use of Neumann boundary conditions.

2.4 Simple Verifications

We now illustrate that the Euler-Lagrange (12), and its corresponding gradient descent flow (16), are the extension for implicit targets of common equations derived in the literature for explicitly represented manifolds. We also explicitly show that the flow equation guarantees, as expected from the derivation above, that if the initial datum is on the target manifold, it will remain on it. We also express the second fundamental form of a manifold that is implicitly represented. All these results will help to further illustrate the approach and verify its correctness.

[7] The intrinsic barycenter G of the map $\vec{u} : \Omega \to \mathcal{N}$ is defined by $G = argmin_{p \in \mathcal{N}} \frac{1}{2} \int_\Omega d^2_{\mathcal{N}}(p, \vec{u}(x))\, dx$. See [10] for more details on the barycenter.

Geodesics as Harmonic Maps

It is well known, see [15, 16, 36], that arc-length parameterized geodesics on the manifold $\mathcal{N}$ satisfy the harmonic maps PDE. If we assume isotropic and homogeneous metric over $\mathcal{N}$, we end up having that (arc-length parameterized) geodesics must satisfy

$$\ddot{\gamma} + \mathbf{H}_{\psi}\left[\dot{\gamma}, \dot{\gamma}\right] \nabla\psi(\gamma) = 0. \tag{21}$$

Liquid Crystals

One of the most popular examples of harmonic maps is given when the target manifold $\mathcal{N}$ is an hyper-sphere. That is, the map is onto S^{d-1}. In this case, the embedding (signed distance) function is simply $\psi(\vec{y}) = \|\vec{y}\| - 1$, $\vec{y} \in I\!R^d$. From this, $\nabla\psi(\vec{y}) = \frac{\vec{y}}{\|\vec{y}\|}$ and $\left(\mathbf{H}_{\psi}(\vec{y})\right)_{ij} = \frac{\delta_{ij}}{\|\vec{y}\|} - \frac{y_i y_j}{\|\vec{y}\|^3}$. We also have that $\mathbf{H}_{\psi}(\vec{u}(x))\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right] = \delta_{ij}\frac{\partial u^i}{\partial x_k}\frac{\partial u^j}{\partial x_k} - \frac{\partial u^i}{\partial x_k}\frac{\partial u^j}{\partial x_k}u_i u_j$, since $\|\vec{u}\| = 1$. In addition, $u^i\frac{\partial u^i}{\partial x_k} = 0$, fact simply obtained taking derivatives with respect to x_k. We then obtain that $\frac{\partial u^i}{\partial x_k}\frac{\partial u^j}{\partial x_k}u_i u_j = \left(\frac{\partial u^i}{\partial x_k}u_i\right)^2 = 0$, and $\sum_{k=1}^{d}\mathbf{H}_{\psi}(\vec{u}(x))\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right] = \sum_{ik}\left(\frac{\partial u^i}{\partial x_k}\right)^2 = \|\mathbf{J}_{\vec{u}}(x)\|^2_{\mathcal{F}}$. Therefore, the corresponding diffusion equation from (16) is

$$\frac{\partial\vec{u}}{\partial t} = \Delta\vec{u} + \|\mathbf{J}_{\vec{u}}\|^2_{\mathcal{F}}\ \vec{u}$$

which is exactly the well known gradient descent flow for this case.

Mapping Restriction onto the Zero Level-Set

We now explicitly show that if the initial datum belongs to the target surface given by the zero level-set of ψ, then the solution to the diffusion flow (16) also belongs to this level-set. This further shows the correctness of our approach.

We basically need to show that $\psi(\vec{u}(x,t)) = 0,\ \forall\, x \in \mathcal{M},\ \forall\, t \geq 0$. If the initial datum is on $\{\psi = 0\}$, then this property is true for $t = 0$. Let's define $\nu(x,t) = \psi(\vec{u}(x,t))$. Then[8]

$$\frac{\partial\nu}{\partial t} = \nabla\psi(\vec{u})\cdot\frac{\partial\vec{u}}{\partial t} = \Delta\vec{u}\cdot\nabla\psi(\vec{u}) + \sum_{k=1}^{d}\mathbf{H}_{\psi}(\vec{u})\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]$$

since ψ is a distance function. In addition, $\frac{\partial\nu}{\partial x_i} = \nabla\psi(\vec{u})\cdot\frac{\partial\vec{u}}{\partial x_i}$, and then

$$\frac{\partial^2\nu}{\partial x_i^2} = \left(\mathbf{H}_{\psi}(\vec{u})\frac{\partial\vec{u}}{\partial x_i}\right)\cdot\frac{\partial\vec{u}}{\partial x_i} + \nabla\psi(\vec{u})\cdot\frac{\partial^2\vec{u}}{\partial x_i^2}.$$

Adding on $i = 1, ..., d$, it follows that $\frac{\partial\nu}{\partial t} = \Delta\nu$, meaning that ν verifies the heat flow. In addition to this, $\frac{\partial\nu}{\partial\mathbf{n}}|_{\partial\mathcal{M}} = \nabla_x\left(\psi(\vec{u}(x,t))\right)\cdot\mathbf{n} = \mathbf{J}_{\vec{u}}^T\nabla\psi(\vec{u})\cdot\mathbf{n} = \left(\nabla\psi(\vec{u})\right)^T\mathbf{J}_{\vec{u}}\mathbf{n} = \left(\nabla\psi(\vec{u})\right)^T\mathbf{0} = 0$, due to the boundary conditions on the evolution of $\vec{u}$.

We have then obtained that ν verifies the heat flow with *Neumann* boundary conditions and with zero initial data. From the uniqueness of the solution, it follows that $\nu(x,t) = 0\ \forall\, x \in \mathcal{M},\ \forall\, t \geq 0$.

[8] All the calculations that follow don't take into account that ψ might fail to be differentiable at some points. This could be addressed by a regularization argument.

Second Fundamental Form for Implicit Surfaces

If we compare the gradient descent flow (and Euler-Lagrange equation) we have obtained with the classical one from harmonic maps, we see that the main difference is that the Christoffel symbols of the target manifold term appearing in the classical formulation has been replaced by a new term that includes the Hessian of the embedding function. We obtained this by first embedding the target manifold and then restricting the search for the minimizing map to the class of maps onto the zero level-set of the embedding function. This approach can be followed to apply this framework to any variational problem. We now show how the same equation can be obtained by simply substituting the second fundamental form of the explicit target manifold by the corresponding expression for an implicit target manifold. This will illustrate how to apply our framework to general PDE's, not necessarily gradient descent flow. The basic idea is just to replace all the PDE components concerning the target manifold by their counterparts for implicit representations.

In [28] (page 150) it is shown that the *scalar second fundamental form* h at a point p of an hypersurface $\mathcal{S}$ can be written in the form

$$h(p)(V,W) = \frac{\mathbf{H}_{\psi}(p)[V,W]}{\|\nabla\psi\|^2}$$

for $V, W \in T_p\mathcal{S}$. According to [28] (page 139) the *vectorial second fundamental form* is given by

$$I\!I(p)(V,W) = h(p)(V,W)\,\frac{\nabla\psi}{\|\nabla\psi\|}$$

From (15) and what we have just seen it is obvious that the *implicit* version of the *harmonic map* Euler-Lagrange is (12).

As stated before, and following the formulas above, the implicit representation of the target surface permits then to compute the second fundamental form using differences on Cartesian grids, without the need to develop new numerical techniques on polygonal grids.

From the result just presented, in order to transform a given PDE into its counterpart when the target manifold is represented in implicit form, all what needs to be done is to re-write all the characteristics of the PDE concerning this target manifold in implicit form. For completeness, in Appendix 1 we present basic facts on calculus on implicitly represented surfaces.

2.5 Explicit Derivation of the Diffusion Flow

Here we first proceed in a naïve way to obtain an equivalent formulation of the gradient descent flow that will help in the numerical implementation. We assume we have a family $\{\vec{u}(\vec{x},t)\}_t$ of mappings from Ω to $\mathcal{N}$. For each t we define the *harmonic energy* of a member of the family as

$$E(t) = \frac{1}{2}\int_{\Omega} \|\mathbf{J}_{\vec{u}(\vec{x},t)}\|_F^2 \; dx$$

We then find a variation of the family such that $E(t)$ decreases. To accomplish this we formally differentiate the energy with respect to t. A simple computation yields

$$\dot{E}(t) = -\int_{\Omega} \vec{u}_t \cdot \Delta\vec{u} \; dx$$

Now, since $\vec{u}(\vec{x},t) \in \mathcal{N} \;\forall\, \vec{x} \in \Omega$ and $\forall\; t$ of smooth existence, one must have $\vec{u}_t(\vec{x},t) \in T_{\vec{u}(\vec{x},t)}\mathcal{N}$. An appropriate choice for $\vec{u}_t$ would be

$$\vec{u}_t \;=\; \mathbf{\Pi}_{T_{\vec{u}(\vec{x},t)}\mathcal{N}}(\Delta\vec{u}) \tag{22}$$

since this makes $\dot{E}(t) = -\int_\Omega \|\vec{u}_t\|^2\, dx \leq 0$.

The projection operator in (22), as we already know (see Appendix 1), can be expressed in a very simple form using ψ (the signed distance function to $\mathcal{N}$),

$$\mathbf{\Pi}_{T_{\vec{p}}\mathcal{N}}(\vec{v}) = \vec{v} - \vec{v}\cdot\nabla\psi(p)\,\nabla\psi(p)$$

Now, it should happen that (22) is *equivalent* to (16). We show this in §7.

2.6 Remarks on the Solutions of the Diffusion Flow

The well posedness of the diffusion problem with *Neumann* boundary conditions is addressed in [22, 32], where the following results are obtained, here included for completeness:

Theorem 1 *For a given C^∞ mapping $\vec{u}_0 : \mathcal{M} \to \mathcal{N} \subset \mathbb{R}^{n+1}$ with $\frac{\partial \vec{u}_0}{\partial \mathbf{n}} = 0$ on $\partial\mathcal{M}$ and for every $2 + dim(\mathcal{M}) < p < +\infty$ there exists an $\epsilon > 0$ (depending on $\vec{u}_0$) and a mapping $\vec{u} : \mathcal{M}\times[0,\epsilon) \to \mathcal{N}$ of class $\mathbf{L}^p_2(\mathcal{M}\times[0,\epsilon], \mathbb{R}^{n+1})$.* [9] *Moreover, $\vec{u}$ is unique and C^∞ except at the corner $\partial\mathcal{M}\times\{0\}$.*

Theorem 2 *Let $(\mathcal{M}, g)$ and $(\mathcal{N}, h)$ be compact Riemmanian Manifolds with convex boundary. Let $\vec{u} : \mathcal{M}\times[0,\omega) \to \mathcal{N}$ be a maximal solution of the diffusion problem with initial value a C^∞ mapping $\vec{u}_0$,*[10] *with $\chi_0 \stackrel{\triangle}{=} \|e[\vec{u}_0]\|_{\mathbf{L}^\infty} > 0$. Let $r \in \mathbb{R}$ be such that $Ric_\mathcal{M} \geq -\frac{r}{2}\, g$,*[11] *and $R \geq 0$ such that all sectional curvatures of $\mathcal{N}$ are not greater than $\frac{R}{4}$. Then,*

1. *In the case $r + R\chi_0 > 0$*
   - *(a) if $R > 0$ then* $\begin{cases} \omega \geq \frac{1}{r}\log(1 + \frac{r}{R\chi_0}) & \text{when } r \neq 0 \\ \omega \geq \frac{1}{R\chi_0} & \text{when } r = 0 \end{cases}$
   - *(b) if $R = 0$ then $\omega = +\infty$.*
2. *In the case $r + R\chi_0 \leq 0$, $\omega = +\infty$.*

3 Maps onto Open Surfaces

So far, we have only addressed the case when the target surface is closed (zero level-set). In this section we briefly deal with open surfaces. We show that when the function is evolving according to the flow in §2.3, the set $\mathcal{C}(t) \stackrel{\triangle}{=} \{\vec{u}(x,t),\ x \in \mathcal{M}\}$ remains inside the initial *convex-hull* of $\mathcal{C}_0 \stackrel{\triangle}{=} \{\vec{u}_0(x),\ x \in \mathcal{M}\}$, $\forall\, t \geq 0$. This property is basically a consequence of the maximum principle. Numerically, this might of course be violated due to numerical errors, and we will later discuss how to correct for this as well.

[9] $\mathbf{L}^p_2(\mathcal{M}\times[0,\epsilon], \mathbb{R}^{n+1})$ is the space of functions $f : \mathcal{M} \to \mathbb{R}^{n+1}$ such that for every $i = 1, \ldots, n+1$, $\nabla_\mathcal{M} f^i$, $\mathbf{H}^\mathcal{M}_{f^i}$ and $\frac{\partial f^i}{\partial t}$ are all in $\mathbf{L}^2(\mathcal{M}\times[0,\epsilon])$.

[10] A solution $\vec{u} : \mathcal{M}\times[0,\omega) \to \mathcal{N}$ of the diffusion problem is *maximal* if it cannot be extended to be a solution on $\mathcal{M}\times[0,\omega+\epsilon)$ for any $\epsilon > 0$ or if $\omega = +\infty$.

[11] $Ric_\mathcal{M}$ stands for the Ricci curvature tensor of $\mathcal{M}$.

3.1 Motivation: The Planar Case

Assume that the target manifold $\mathcal{N}$ is flat, for example R^k (we still assume that the domain manifold $\mathcal{M}$ is flat). Let $\vec{u}(x,t)$ solve $\frac{\partial \vec{u}}{\partial t} = \Delta \vec{u}$ for $x \in \mathcal{M}$ and $t \geq 0$, and $\frac{\partial \vec{u}}{\partial \mathbf{n}}\Big|_{\partial \mathcal{M}} = 0$. Let Ξ be a convex set of R^k with smooth boundary (this guarantees that the distance function is also smooth almost everywhere, see [36] for a formal statement), and ξ the signed distance function to this set (positive outside and negative inside). Define $g(x,t) \stackrel{\triangle}{=} \xi(\vec{u}(x,t))$. It then immediately follows that $\frac{\partial g}{\partial t} - \Delta g = -\sum_{i=1}^{k} \mathbf{H}_{\xi}(\frac{\partial \vec{u}}{\partial x_k}, \frac{\partial \vec{u}}{\partial x_k})$.[12] Since Ξ is convex, so it is ξ. Then, the *Hessian* of ξ is *positive semi-definite*, meaning that $\frac{\partial g}{\partial t} - \Delta g \leq 0$. Following the scalar maximum principle, $\max_{\{x \in \mathcal{M},\, t \geq 0\}} g(x,t) = \max_{\{x \in \mathcal{M}\}} g(x,0)$. If $\{\vec{u}_0(x),\, x \in \mathcal{M}\} \subseteq \Xi$, which is equivalent to $0 \geq \xi(\vec{u}_0(x)) = g(x,0)$, we obtain that $g(x,t) \leq 0$, and $\vec{u}(x,t) \in \Xi$, for all $x \in \mathcal{M}$ y $t \geq 0$.

3.2 The General Case

The main result presented below is from [22]. We quote it here for completeness.[13]

Theorem 3 *Let $\vec{u}(x,t)$ be the solution of (16) at time t. Let us assume that for $t \leq T$ this solution remains smooth. Let $I_0 = \vec{u}_0(\Omega)$, and $\mathcal{I}_0$ be the* convex hull *of I_0. Then for $(x,t) \in \Omega \times [0,T]$, $\vec{u}(x,t) \in \mathcal{I}_0$.*

4 Maps onto Implicit Submanifolds

Here we present a modification to the diffusion flow previously presented suited to diffuse data that belongs to a certain *submanifold* $\mathcal{C}$ of $\mathcal{N} = \{\psi = 0\}$. We specify the submanifold by $\{\psi = 0\} \cap \{\Phi = 0\}$, where we select $\Phi : I\!R^N \to R$ to be the signed *intrinsic* (to $\mathcal{N}$) distance function to $\{\Phi = 0\}$, satisfying (see Appendix 1 for the notation)

$$1 = \|\nabla_{\psi} \Phi\| = \sqrt{\|\nabla \Phi\|^2 - |\nabla \psi \cdot \nabla \Phi|^2} \tag{23}$$

In addition we specify the condition

$$\Phi(p) = 0 \text{ for } p \in \mathcal{K}_{\Phi}$$

where

$$\mathcal{K}_{\Phi} = \{x \in I\!R^N \,|\, x = p + \alpha \nabla \psi(p), \text{with } p \in \mathcal{C},\, \alpha \in I\!R\}$$

is the *cone* intersecting $\{\psi = 0\}$ at $\mathcal{C}$ and director rays normal also to $\{\psi = 0\}$.

The reason for specifying the submanifold this way is that we cannot proceed as before, simply specifying the submanifold as the zero level set of it's Euclidean distance function. This is because such function would be singular precisely onto the submanifold.

As we show in Appendix 1, the Hessian of Φ, *intrinsic* to $\mathcal{N}$ evaluated at the point p, and restricted to $T_p\mathcal{N}$, can be written in the form

$$\mathbf{H}_{\Phi}^{\mathcal{N}}(p) = \mathbf{H}_{\Phi}(p) - \Lambda(p)\ \mathbf{H}_{\psi}(p) \tag{24}$$

where $\Lambda(p) = \nabla \Phi(p) \cdot \nabla \psi(p)$. This expression will be used below.

[12] Note once again that we are omitting details regarding the correct handling of the distance function, since it is not everywhere differentiable. However, by a regularization argument, the same conclusion holds.

[13] The proof of this result has a lot of interest in itself since it can be carried out within the implicit framework introduced in this paper.

4.1 The Minimization of the Functional

We now derive the Euler-Lagrange corresponding to this additional mapping restriction. For this, we use a technique slightly different that the one in §2.1.

Let us assume that $\vec{u}$ achieves a minimum of the energy functional (1). We must build a variation of $\vec{u}$ that belongs to $\mathcal{C}$, the intersection of the zero level-sets of two embedding functions (and not just of ψ as before). It is clear that one such variation would be

$$\vec{w}_\lambda = \mathbf{\Pi}_{\mathcal{C}} \left(\vec{u} + \lambda \vec{v} \right)$$

We are interested only on those terms of $e[\vec{w}_\lambda]$ that do not vanish after the $\sum_{i=1}^{N} \frac{\partial}{\partial x_i}(\bullet)\Big|_{\lambda=0}$ operation, namely those linear in λ. Therefore we only preserve those terms in $\vec{w}_\lambda$ which are constant or linear in λ:

$$\vec{w}_\lambda \simeq \vec{u} + \lambda \, \mathbf{\Pi}_{T_{\vec{u}}\mathcal{C}}(\vec{v})$$

We write

$$\begin{aligned} \mathbf{\Pi}_{T_{\vec{u}}\mathcal{C}}(\vec{v}) &= \mathbf{\Pi}_{T_{\vec{u}}\{\psi=0\}} \left\{ \; \vec{v} - (\vec{v} \cdot \nabla_\psi \Phi(\vec{u})) \, \nabla_\psi \Phi(\vec{u}) \; \right\} \\ &= \vec{v} - (\vec{v} \cdot \nabla_\psi \Phi(\vec{u})) \, \nabla_\psi \Phi(\vec{u}) - (\vec{v} \cdot \nabla \psi(\vec{u})) \, \nabla \psi(\vec{u}) \end{aligned}$$

where $\nabla_\psi \Phi(\vec{u}) = \nabla \Phi(\vec{u}) - \Lambda(\vec{u}) \; \nabla \psi(\vec{u})$ is the gradient of Φ *intrinsic* to $\{\psi = 0\}$.
In this way we find that (up to a first order in λ):

$$\begin{aligned} e[\vec{w}_\lambda] \;\simeq\; & e[\vec{u}] + \lambda \; \sum_{i=1}^{N} \vec{u}_{x_i} \cdot [\vec{v}_{x_i} - \vec{v}_{x_i} \cdot \nabla_\psi \Phi(\vec{u}) \; \nabla_\psi \Phi(\vec{u}) \\ & - \; \vec{v} \cdot \frac{\partial \nabla_\psi \Phi(\vec{u})}{\partial x_i} \; \nabla_\psi \Phi(\vec{u}) - \vec{v} \cdot \nabla_\psi \Phi(\vec{u}) \; \frac{\partial \nabla_\psi \Phi(\vec{u})}{\partial x_i} \\ & - \; \vec{v}_{x_i} \cdot \nabla \psi(\vec{u}) \; \nabla \psi(\vec{u}) - \vec{v} \cdot \frac{\partial \nabla \psi(\vec{u})}{\partial x_i} \nabla \psi(\vec{u}) - \vec{v} \cdot \nabla \psi(\vec{u}) \frac{\partial \nabla \psi(\vec{u})}{\partial x_i}] \end{aligned} \tag{25}$$

Since $\Phi(\vec{u}) = \psi(\vec{u}) = 0$, differentiating with respect to x_i we obtain that $\nabla \Phi(\vec{u}) \cdot \vec{u}_{x_i} = \nabla \psi(\vec{u}) \cdot \vec{u}_{x_i} = 0$, and therefore

$$\nabla_\psi \Phi(\vec{u}) \cdot \vec{u}_{x_i} = 0$$

The expression (25) can be simplified to obtain

$$e[\vec{w}_\lambda] \simeq e[\vec{u}] + \lambda \; \sum_{i=1}^{N} \vec{u}_{x_i} \cdot \left[\vec{v}_{x_i} - \vec{v} \cdot \nabla_\psi \Phi(\vec{u}) \; \frac{\partial \nabla_\psi \Phi(\vec{u})}{\partial x_i} - \vec{v} \cdot \nabla \psi(\vec{u}) \, \frac{\partial \nabla \psi(\vec{u})}{\partial x_i} \right]$$

Moreover, since

$$\frac{\partial \nabla_\psi \Phi(\vec{u})}{\partial x_i} = \mathbf{H}_\Phi \vec{u}_{x_i} - \frac{\partial \Lambda}{\partial x_i}(\vec{u}) \; \nabla \psi(\vec{u}) - \Lambda(\vec{u}) \; \mathbf{H}_\psi \vec{u_{x_i}}$$

we have

$$\frac{\partial \nabla_\psi \Phi(\vec{u})}{\partial x_i} \cdot \vec{u}_{x_i} = \mathbf{H}_\Phi^{\mathcal{N}}[\vec{u}_{x_i}, \vec{u}_{x_i}]$$

With all this in mind we find that (again, up to first order in λ)

$$e[\vec{w}_\lambda] \simeq e[\vec{u}] + \lambda \sum_{i=1}^{N} \vec{u}_{x_i} \cdot \left[\vec{v}_{x_i} - \vec{v} \cdot \nabla_\psi \Phi(\vec{u})\ \mathbf{H}^{\mathcal{N}}_\Phi[\vec{u}_{x_i}, \vec{u}_{x_i}] - \vec{v} \cdot \nabla\psi(\vec{u})\ \mathbf{H}_\psi(\vec{u})[\vec{u}_{x_i}, \vec{u}_{x_i}]\right]$$

Using this expression, after imposing that $\frac{\partial}{\partial\lambda}\big|_{\lambda=0} \int_\Omega e[\vec{w}_\lambda]\ dv = 0$ for every $\vec{v}$, we find that the Euler-Lagrange is

$$\Delta\vec{u} + \left(\sum_k \mathbf{H}_\psi\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]\right) \nabla\psi(\vec{u}) + \left(\sum_k \mathbf{H}^{\mathcal{N}}_\Phi\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]\right) \nabla_\psi\Phi(\vec{u}) = 0. \tag{26}$$

an expression utterly predictable.

4.1.1 Simple Verification

As for the case of closed manifolds, we now verify that in fact the gradient descent corresponding to the Euler-Lagrange (26) keeps $\vec{u}$ in $\{\psi = 0\} \cap \{\Phi = 0\}$. We just need to show that $\nu(x,t) \stackrel{\triangle}{=} \psi(\vec{u}(x,t))$ and $\mu(x,t) \stackrel{\triangle}{=} \Phi(\vec{u}(x,t))$ are always zero. The idea is the same we used in §2.4, it is enough to show that both ν and μ satisfy the *heat equation* with adiabatic *boundary conditions.*

1. ψ

   We have

   $$\nu_t = \nabla\psi \cdot \Delta\vec{u} + \sum_k \mathbf{H}_\psi\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]$$

   since $\nabla\psi \perp \nabla_\psi\Phi$. Also

   $$\Delta\nu = \nabla\psi \cdot \Delta\vec{u} + \sum_k \mathbf{H}_\psi\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]$$

   and

   $$\nu_t = \Delta\nu$$

2. Φ

   We have

   $$\mu_t = \nabla\Phi \cdot \Delta\vec{u} + \nabla\Phi \cdot \nabla_\psi\Phi\ \left(\sum_k \mathbf{H}^{\mathcal{N}}_\Phi\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]\right) + \Lambda\ \left(\sum_k \mathbf{H}_\psi\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]\right)$$

   From $\nabla\Phi \cdot \nabla_\psi\Phi = \nabla_\psi\Phi \cdot \nabla_\psi\Phi = \|\nabla_\psi\Phi\|^2 = 1$, the above equation continues as

$$
\begin{aligned}
&= \nabla\Phi \cdot \Delta\vec{u} + \left(\sum_k \mathbf{H}_\Phi^{\mathcal{N}}\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]\right) + \Lambda\left(\sum_k \mathbf{H}_\psi\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]\right) \\
&= \nabla\Phi \cdot \Delta\vec{u} + \left(\sum_k \mathbf{H}_\Phi\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]\right)
\end{aligned}
$$

Also

$$
\Delta\mu = \nabla\Phi \cdot \Delta\vec{u} + \left(\sum_k \mathbf{H}_\Phi\left[\frac{\partial\vec{u}}{\partial x_k}, \frac{\partial\vec{u}}{\partial x_k}\right]\right)
$$

and then

$$
\mu_t = \Delta\mu
$$

Finally, it is easy to see that both ν and μ satisfy *Neumann boundary conditions.* Since at $t = 0$ both functions are zero, we must have that they are identically zero.

5 Implicit Domain Manifolds and p-Harmonic Maps

For completeness, we present now the formulas corresponding to the case where both the domain and target manifolds are represented in implicit form (with the implicitizing functions being the corresponding signed distance ones). Deriving these formulas is straightforward using the framework here presented, when combined with the work in [3]. We also show the corresponding flows for p-harmonic maps.

5.1 p-Harmonic Maps

We still assume $\mathcal{M}$ to be planar. The energy density (2) (but no the dependence of the energy on its density) is redefined as follows. For every $p \in [1, +\infty)$ let

$$
e_p[\vec{u}] \stackrel{\triangle}{=} \frac{1}{p}\|\mathbf{J}_{\vec{u}}\|_{\mathcal{F}}^p
$$

A simple application of variational calculus leads to conclude that[14]

$$
\vec{u}_t = p^{1-\frac{2}{p}}\ \mathbf{\Pi}_{\nabla\psi(\vec{u})}\left(\nabla \cdot \left(\left(e_p[\vec{u}]\right)^{1-\frac{2}{p}}\mathbf{J}_{\vec{u}}^T\right)\right) \tag{27}
$$

Note that if $p < 2$ difficulties are expected to arise, see [40] and the references therein.

5.2 Generic (Implicit) Domain Manifolds

Let $\mathcal{M} = \{x \in I\!R^m : \phi(x) = 0\}$, where $\phi(\cdot)$ is the signed distance function to $\mathcal{M}$, then the diffusion is given by:

$$
\vec{u}_t = \nabla \cdot \left(\mathbf{\Pi}_{\nabla\phi}\mathbf{J}_{\vec{u}}^T\right)\ +\ \left(\sum_{k,r} \mathbf{H}_\psi[\vec{u}_{x_r}, \vec{u}_{x_k}]\left(\mathbf{\Pi}_{\nabla\phi}\right)_{kr}\right)\nabla\psi \tag{28}
$$

[14]The divergence operator convention (for a matrix A) we have used is $\nabla \cdot A = \left(\nabla \cdot \vec{A}_{v1}\middle| \ldots \middle|\nabla \cdot \vec{A}_{vr}\right)$, where $\vec{A}_{vi}$ stands for the i-th column of A. That is, we apply a columnwise divergence.

The whole deduction rests upon the redefinition of the energy (1) and its density (2). Now we should define the energy density to be

$$e_\phi[\vec{u}] \stackrel{\triangle}{=} \frac{1}{2}\|\mathbf{J}_{\vec{u}}^{\phi}\|_{\mathcal{F}}^2$$

where the *intrinsic Jacobian* of $\vec{u}$ can be written as (see Appendix 1 for more details) $\mathbf{J}_{\vec{u}}^{\phi} = \mathbf{J}_{\vec{u}}\,\mathbf{\Pi}_{\nabla\phi}$.

The new definition for the energy should be:[15]

$$E[\vec{u}] \stackrel{\triangle}{=} \int_{I\!R^m} e_\phi[\vec{u}]\,\delta(\phi(x))\,dx \tag{29}$$

Comparing this with (15), we can infer the implicit form of the Christoffel symbols:[16]

$$\mathbf{\Gamma}_{ij}^{l}(\vec{u}) = \frac{\partial^2\psi}{\partial u^i \partial u^j}(\vec{u})\ \frac{\partial\psi}{\partial u^l}(\vec{u})$$

5.3 Generic (Implicit) Domain Manifold and p-Harmonic Maps

Using both generalization presented above, we arrive at the following formula with a bit more computational effort

$$\vec{u}_t = p^{1-\frac{2}{p}}\ \mathbf{\Pi}_{\nabla\psi(\vec{u})}\left(\nabla\cdot\left(\left(e_{\phi,p}[\vec{u}]\right)^{1-\frac{2}{p}}\ \mathbf{\Pi}_{\nabla\phi}\mathbf{J}_{\vec{u}}^{T}\right)\right) \tag{30}$$

where

$$e_{\phi,p}[\vec{u}] \stackrel{\triangle}{=} \frac{1}{p}\|\mathbf{J}_{\vec{u}}^{\phi}\|_{\mathcal{F}}^p$$

6 Diffusion of Tangent and Normal Directions

Throughout this section we will assume $dim(\mathcal{M}) = dim(\mathcal{N})$. Assume we want to diffuse intrinsic vectorial data constrained to be a direction (unit norm) and to be either normal or tangent to the domain manifold. We can then minimize the functional (29) taking a variation of the form (assume $\vec{u}$ minimizes the energy functional while satisfying both $\|\vec{u}\| = 1$ and $\Pi(\vec{u}) = \vec{u}$)

$$\vec{u}_\lambda(x) \stackrel{\triangle}{=} \frac{\vec{u} + \lambda\Pi(\vec{v})}{\|\vec{u} + \lambda\Pi(\vec{v})\|}$$

where $\vec{v} : \mathcal{M} \to I\!R^d$ is smooth and Π is either $\mathbf{\Pi}_{T_x\mathcal{M}}$ or $\mathbf{\Pi}_{N_x\mathcal{M}}$ (projection onto the tangent or normal space respectively). Let $\vec{w} = \Pi(\vec{v})$, then it follows easily that

$$\left.\frac{dE[\vec{u}_\lambda]}{dt}\right|_{\lambda=0} = -\int_{I\!R^m}\left\{\Delta_\phi\vec{u}\ +\ 2\,e_\phi[\vec{u}]\ \vec{u}\right\}\cdot\vec{w}\ \ \delta(\phi(x))\,dx$$

Imposing $\left.\frac{dE[\vec{u}_\lambda]}{dt}\right|_{\lambda=0} = 0$ for all v implies

$$\Pi\left(\Delta_\phi\vec{u}\ +\ 2\,e_\phi[\vec{u}]\ \vec{u}\right) = \Pi\left(\Delta_\phi\vec{u}\right)\ +\ 2\,e_\phi[\vec{u}]\ \vec{u} = 0$$

[15] We have already taken into account that $\|\nabla\phi\| = 1$.

[16] Of course $g^{ij} = \left(\mathbf{\Pi}_{\nabla\phi}\right)_{ij}\ \ (= g_{ij}^{-1})$. Then, it is nice to observe (although formally incorrect) that since $\mathbf{\Pi}_{\nabla\phi}\nabla\phi = 0$, then the metric $g : I\!R^d \to I\!R^{d\times d}$ has eigenvalue $+\infty$ in the direction given by $\nabla\phi$ thus prohibiting intermingling of information between adjacent level sets of ϕ.

Finally, the diffusion flow obtained is

$$\frac{\partial \vec{u}}{\partial t}(x,t) = \Pi_x \left(\Delta_\phi \vec{u}(x,t)\right) \; + \; 2\, e_\phi[\vec{u}](x,t) \; \vec{u}(x,t) \tag{31}$$

Note that if the PDE (31) admits a smooth solution until time T, and if (for instance) we are dealing with tangent directions diffusion, the function $f(x,t) \stackrel{\triangle}{=} \nabla\phi(x) \cdot \vec{u}(x,t)$ satisfies $f_t(x,t) = 2e_\phi[\vec{u}]f(x,t)$. Therefore

$$f(x,t) = f(x,0)e^{2\int_0^t e[\vec{u}](x,t)\,dt}$$

thus verifying that if $\nabla\phi(x) \cdot \vec{u}(x,0) = 0$ then $\nabla\phi(x) \cdot \vec{u}(x,t) = 0$ for $t \leq T$. We also want to check whether $\|\vec{u}(x,t)\| = 1 \; \forall \; (x,t)$. Let

$$F_\phi[\vec{u}](t) \stackrel{\triangle}{=} \frac{1}{2} \int_{I\!R^d} \|\mathbf{J}_{\vec{u}}\|^2_{\mathcal{F}} \; \delta(\phi(x)) \; dx$$

then $\dot{F}_\phi[\vec{u}](t) = -\int_{I\!R^d} \vec{u}_t \cdot \Delta_\phi \vec{u} \;\; \delta(\phi(x))\, dx$. Since both $\|\vec{u}\| = 1$ and $\Pi(\vec{u}) = \vec{u}$ (so $\Pi(\vec{u}_t) = \vec{u}_t$ since Π does not depend on t) must hold, and in order to make $\dot{F}_\phi[\vec{u}](t)$ non-positive we choose

$$\vec{u}_t = \Pi\, \mathbf{\Pi}_{T_{\vec{u}}\{\|\vec{u}\|=c\}} \Delta_\phi \vec{u} \tag{32}$$

where $\mathbf{\Pi}_{T_{\vec{u}}\{\|\vec{u}\|=c\}} = \mathbf{I} - \frac{\vec{u}}{\|\vec{u}\|}\frac{\vec{u}^T}{\|\vec{u}\|}$ for any $c > 0$.

Note that the above evolution indeed forces $\vec{u}(x,t)$ to satisfy both imposed conditions. Let $\vec{v} : I\!R^d \to I\!R^d$ be such that $\Pi(\vec{v}) = \vec{0}$ then $(\vec{v} \cdot \vec{u})_t = \vec{v} \cdot \vec{u}_t = \vec{v} \cdot \Pi\, \mathbf{\Pi}_{T_{\vec{u}}S^{d-1}} \Delta_\phi \vec{u} = \Pi^T \vec{v} \cdot \mathbf{\Pi}_{T_{\vec{u}}\{\|\vec{u}\|=c\}} \Delta_\phi \vec{u} = 0$, since the projection matrix is symmetric, and just using this we have $\left(\frac{1}{2}\|\vec{u}\|^2\right)_t = \vec{u} \cdot \vec{u}_t = \vec{u} \cdot \mathbf{\Pi}_{T_{\vec{u}}\{\|\vec{u}\|=c\}} = 0$ trivially. Finally, using $\|\vec{u}\| = 1$ and carrying out some computations in a way similar to §7 below,[17] one can prove that (32) reduces to (31).

7 Numerical Implementation

We now discuss the numerical implementation of the flows previously introduced. Since the target manifold is now implicitly represented, we can basically use classical numerical techniques on Cartesian grids. Although as we have shown, the flows guarantee that the map remains on the target (sub-)manifold, numerical errors can move it away from it, requiring a simple projection step.

When dealing with submanifolds, although the evolution equations also guarantee that the solution will remain inside the convex hull, once again due to numerical discretization $\vec{u}$ could be taken outside of it during the evolution. In order to numerically project it back, we need to have a distance function to this convex hull defined on the implicitly defined target manifold. In [30] we have shown how to computationally optimal compute such a distance function on implicitly defined manifolds, and this is the technique used for this projection into the convex hull.

An explicit scheme can be devised to implement (28). However it turns out that it is more convenient to implement a *mathematically equivalent* evolution, as shown in [12]. More specifically, the *equivalent* evolution is

$$\frac{\partial u}{\partial t} = \Delta u \; - \; (\Delta u \cdot \nabla\psi)\nabla\psi \tag{33}$$

That both evolutions are equivalent is easy to see, and we show this next.

[17] The main difference is that now one must take into account the Laplace-Beltrami expressed "implicitly," see Appendix 1 for more details on intrinsic differential operators within the implicit framework.

One has that $f(x,t) \stackrel{\triangle}{=} \psi(\vec{u}(x,t)) = 0 \; \forall (x,t) \in \Omega \times I\!R^+ \cup \{0\}$ for $\vec{u}(\cdot,\cdot)$ satisfying (16). Now, differentiating f with respect to x_i we obtain

$$\nabla\psi(\vec{u}) \cdot \vec{u}_{x_i} = 0$$

Differentiating again with respect to x_i,

$$\mathbf{H}_\psi[\vec{u}_{x_i}, \vec{u}_{x_i}] \; + \; \nabla\psi \cdot \vec{u}_{x_i x_i} = 0$$

Summing for all i,

$$\sum_i \mathbf{H}_\psi[\vec{u}_{x_i}, \vec{u}_{x_i}] \; + \; \nabla\psi \cdot \Delta\vec{u} = 0$$

and using the previous expression we derive (33) from (16).

7.1 Numerical Scheme

All the coding was done using **Flujos** as the main core (see [19]) and **VTK** (see [49]) for visualization purposes. All the examples below were carried based in equation (30). Its numerical implementation is straightforward (at least when $p = 2$). We used forward time discretization (explicit scheme), and for the spatial discretization, we used the following well known recipe. To spatially discretize

$$f_t(x,t) = \nabla \cdot (\mathbf{K}(x)\nabla f(x,t)) \tag{34}$$

($\mathbf{K}(x)$ is a symmetric positive semi-definite matrix), we consider *backward* approximation of the divergence and a *forward* approximation of the gradient. Let's explain how this applies in our situation, and for that we assume $p = 2$ in (30). Then the equation we have to implement is

$$\vec{u}_t(x,t) = \mathbf{\Pi}_{\nabla\psi(\vec{u}(x,t))} \left(\nabla \cdot \left(\mathbf{\Pi}_{\nabla\phi(x)} \left(\mathbf{J}^T_{\vec{u}}(x,t) \right) \right) \right)$$

If we don't take into account the outer projection matrix, every coordinate of $\vec{u}$ evolves according to

$$u^i_t(x,t) = \nabla \cdot \left(\mathbf{\Pi}_{\nabla\phi(x)} \nabla u^i(x,t) \right)$$

having for each component the same structure than the model evolution (34). We then borrow the above discretization for our evolution. If we consider the coupling among different u^i's imposed by the projection matrix $\mathbf{\Pi}_{\nabla\psi(\cdot)}$, we see that we still preserve numerical stability since this matrix is positive semidefinite and has spectral radius not greater than 1.[18] In more detail, it can be shown after some calculations (see [21, 42]) that for the scheme (p now denotes a position over the grid)

$$\vec{v}_p^{n+1} = \vec{v}_p^n + \Delta t \; \mathbf{P}(\vec{v}_p^n)(\nabla^- \cdot \left(\mathbf{Q}(p)\nabla^+ \vec{v}_p^n \right))$$

the stability condition is of the form ($\lambda = \frac{\Delta t}{(\Delta x)^2}$)

$$\lambda \leq \min_{p,u} \frac{S(p)}{\rho(\mathbf{P}(u)) \max\{S^2(p), D^2(p)\}}$$

or

[18] Note that $\|\vec{v}\|^2 \geq \mathbf{\Pi}_{\nabla\psi}[\vec{v},\vec{v}] = \|v\|^2 - |\nabla\psi \cdot \vec{v}|^2 \geq 0$ for all $\vec{v}$. We have used that ϕ is a distance function.

$$\lambda \leq \frac{1}{\max_u \rho(\mathbf{P}(u))} \min_p \left\{ \frac{S(p)}{\max\{S^2(p), D^2(p)\}} \right\}$$

where $\rho(\mathbf{P}(p))$ stands for the spectral radius of the matrix $\mathbf{P}(p)$, $S(p) = \sum_{ij} (q_{ij}(p) + q_{ij}(p - \Delta x \vec{e}_i))$, and $D(p) = \sum_{ij} (q_{ij}(p) - q_{ij}(p - \Delta x \vec{e}_i))$. In our case we may admit $D(\cdot)$ to be small compared with $S(\cdot)$ (given the identification $\mathbf{Q} \leftrightarrow \mathbf{\Pi}_{\nabla\phi}$) when Δx is small. This can be easily related to the curvatures of $\{\phi = 0\}$ giving a condition on the sampling of the distance function (ϕ) representing the domain manifold. This condition mainly means that we require a fine enough sampling as to guarantee that the change in the normals to the level surfaces of ϕ is small between adjacent grid points. This condition is obviated when the domain manifold is planar. So the stability condition becomes

$$\lambda \leq \frac{1}{\max_u \rho(\mathbf{P}(u)) \ \max_p S(p)}$$

Since by Cauchy-Schwartz's inequality (and the aforementioned assumption on the change of $\nabla\phi$ between adjacent grid points) $2\,d$ (in practise) upper-bounds $S(p)$, remembering the fact that $\rho(\mathbf{P}(p)) \leq 1$, we arrive at $\lambda \leq \frac{1}{2\,d}$. Note that if a more careful implementation is desired, good choices are ADI or AOS schemes, see [50].

All derivatives in $\mathbf{\Pi}_{\nabla\psi(\cdot)}$ and $\mathbf{\Pi}_{\nabla\phi(\cdot)}$ were approximated by central differences. An interpolation scheme had to be used since the evaluations of $\mathbf{\Pi}_{\nabla\psi(\cdot)}$ in the above equation are at positions given by $\vec{u}(x,t)$, positions not necessarily on the underlying grid. We used linear interpolation for this purpose.

Note that as done in [3], when the domain manifold is also implicitly represented, the values of the map on it are periodically extended to its surrounding offset due to stability considerations. Also, as explained before, due to numerical discretization, the discretely computed solution map can be taken out of the target manifold during the evolution. In this paper, we simply project it back at every iteration. We have seen that this projection is a trivial step due to the fact that the embedding is a distance function. It is quite straightforward to show that the results reported in [1] can be extended for our equations as well, at least for convex hyper-surfaces.

7.2 Numerical Examples

In all the examples below, the domain manifold $\mathcal{M}$ is either the Euclidean space $\mathbb{R}^2$ or an implicit torus. The target manifold $\mathcal{N}$ is an implicit surface in $\mathbb{R}^3$, that is, the zero level-set of $\psi : \mathbb{R}^3 \to \mathbb{R}$, ψ being a signed distance function (this is of course also the case when the surface is a sphere, ψ being as in §2.4).

In order to present interesting examples we construct texture maps, add noise to them, and then diffuse them using our framework. Let $\mathcal{S}$ be the surface onto which we want to map a given (planar) image defined in a subset $D \subset \mathbb{R}^2$. Then the *texture pap* is a map $\vec{T} : \mathcal{S} \to D$. Once the map is known, we inverted it to find a map $\vec{u}_0 : D \to \mathcal{S}$. Then, we built up the noisy map $\vec{u} : D \to \mathcal{S}$ defined by

$$\vec{u}(x) = \mathbf{\Pi}_{\mathcal{S}} \left(\vec{u}_0(x) + \vec{\mathbf{n}}(x) \right)$$

where $\vec{\mathbf{n}} : D \to \mathcal{S}$ is random map with small prescribed power σ. We then feed the evolution (16) with $\vec{u}$ as initial condition, and Neumann boundary conditions. After a certain number of steps we stop the evolution, invert the resulting map, and use it as a texture map to paint the surface with a certain texture.[19]

As a means of finding a suitable $\vec{T}$ we have extended the work in [47] (a multidimensional scaling approach), combined with the technique developed in [30] for computing distances on implicit surfaces.

[19] Note that we are not proposing this as a complete texture mapping alternative, it is just to provide an illustrative example.

Figure 1: Diffusion of a noisy texture map (left) onto an implicit sphere (right). *(This is a color figure.)*

In all the steps just described there are some minor implementation details, mainly regarding interpolation tasks, that we omit for the sake of clarity.

In Figures 1, 2 and 3 we then denoise vectors from the plane $\mathbb{R}^2$ to a 3D surface defined as the zero level-set of $\psi : \mathbb{R}^3 \to \mathbb{R}$ and map a texture image to the surface using the obtained map. Note that the map is the one being processed, not the image itself.

We also show an example of diffusion of random maps from an implicit torus to the implicit bunny model, see Figure 4. As expected from the theory, when evolving this set with the harmonic flow, the set converges to a unique point.

8 Conclusions

In this paper we have shown how to implement variational problems and partial differential equations onto general target surfaces. We have also addressed the case of open target surfaces and sub-manifolds. The key concept is to represent the target (sub-)manifolds in implicit form, and then implement the equations in the corresponding embedding space. This framework completes the work with general domain manifolds reported in [3], thereby providing a complete solution to the computation of maps between generic manifolds.

Acknowledgments

We thank Alberto Bartesaghi, Marcelo Bertalmio and Robert Gulliver for interesting conversations during this work and also Alvaro Pardo for his careful reading of the paper. FM performed part of this work while visiting the University of Minnesota. This work was partially supported by a grant from the Office of Naval Research ONR-N00014-97-1-0509, the Office of Naval Research Young Investigator Award, the Presidential Early Career Awards for Scientists and Engineers (PECASE), a National Science Foundation CAREER Award, and the National Science Foundation Learning and Intelligent Systems Program (LIS).

Figure 2: Diffusion of a noisy texture map onto an implicit teapot. We show two different views (noisy on the top and regularized on the bottom). *(This is a color figure.)*

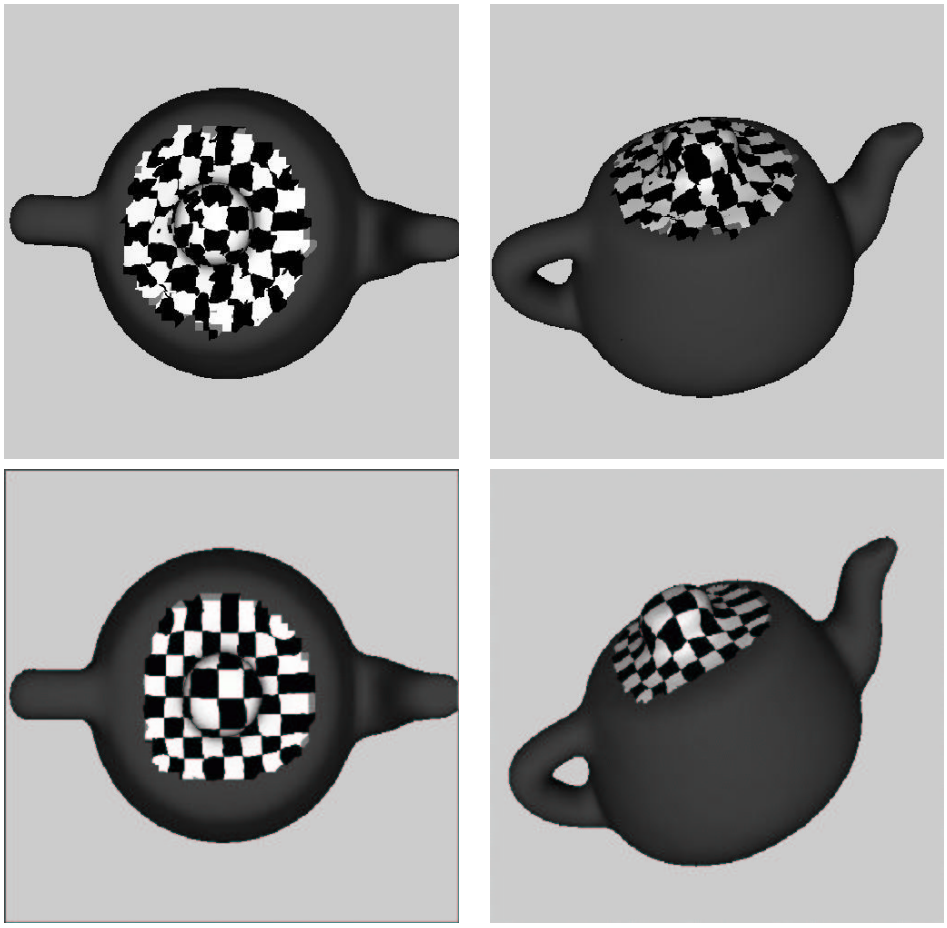


Figure 3: Diffusion of a texture map for an implicit teapot (noisy on he top and regularized on the bottom). A chess board texture is mapped. *(This is a color figure.)*

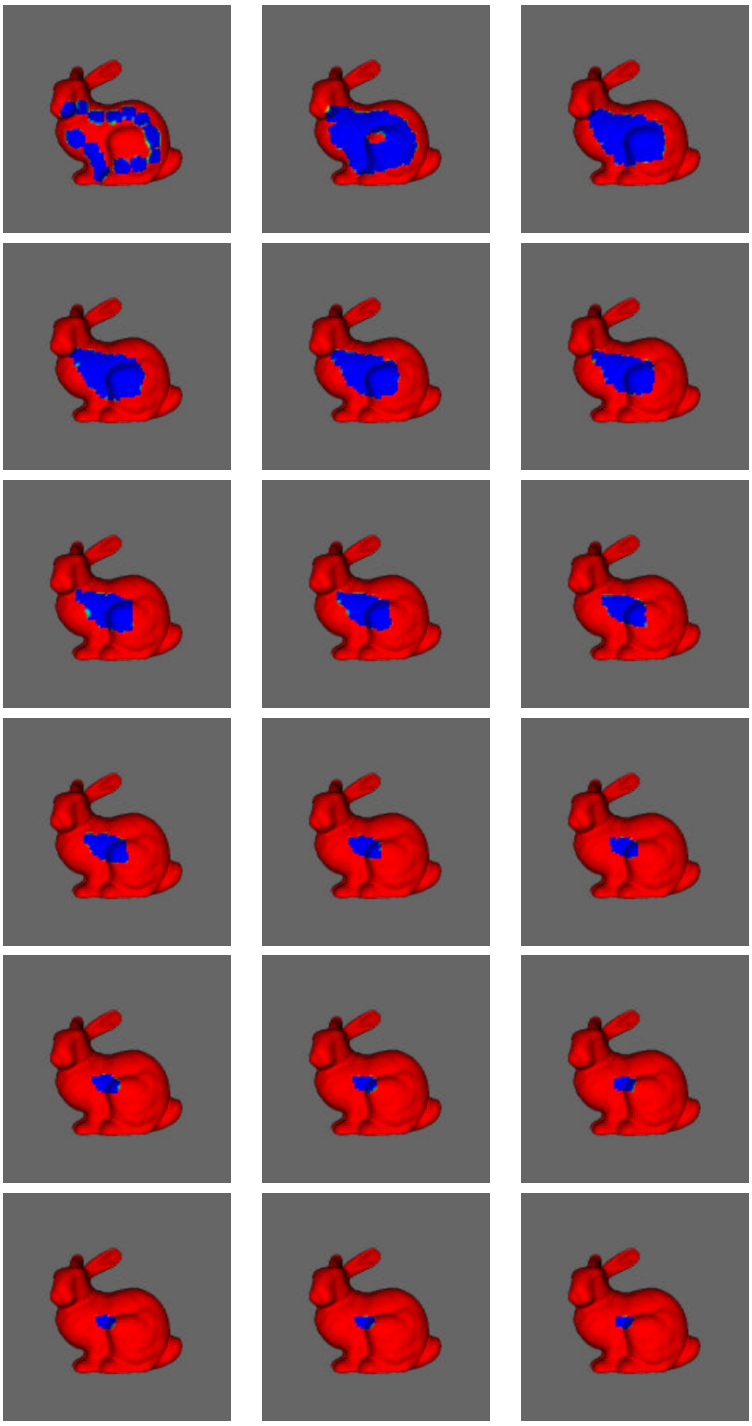


Figure 4: Diffusion of a random map from an implicit torus to the implicit bunny. In blue are marked those points of the bunny's surface pointed by the map at every instant. Different figures correspond to increasing instances of the evolution, from top to bottom and left to tight. We show the map at 17 of 100 iterations performed to the initial map with a time step of .01. We used the 2-harmonic heat flow with adiabatic conditions. *(This is a color figure.)*

Appendix 1: Implicit Calculus

We now present basic facts about differential calculus on implicitly represented surfaces. For more information see for example [2, 8, 29].

We have a smooth scalar function $f : I\!R^d \to I\!R$, and a smooth vector field $\vec{\lambda} : I\!R^d \to I\!R^D$ (d and D are not necessarily equal). The manifold onto which the calculus is to be done is represented as $\mathcal{S} = \{\psi = 0\}$, for $\psi(\cdot)$ the signed distance function to $\mathcal{S}$.

All the ideas of differentiation can be obtained from simple considerations related to the restriction of the function to a geodesic curve living in the manifold. We consider an arc-length parameterized geodesic curve $\gamma : [-\epsilon, \epsilon] \to \mathcal{S}$ such that $\gamma(0) = p$ is a given point of $\mathcal{S}$. We denote $F(t) = f(\gamma(t))$ and $\vec{\Lambda}(t) = \vec{\lambda}(\gamma(t))$.

Implicit gradient

We differentiate once $F(t)$ to obtain $\dot{F}(0) = \nabla f(p) \cdot \dot{\gamma}(0)$. Since $\dot{\gamma}(0) \in T_p\mathcal{S}$ (the tangent plane), we find the implicit gradient of f at p to be $\nabla_{\mathcal{S}} f(p) = \nabla f(p) - \nabla f(p) \cdot \vec{n}(p)\ \vec{n}(p)$, where $\vec{n}(p)$ stands for the normal to the manifold at p. Since we can also write $\vec{n}(p) = \nabla\psi(p)$, we obtain

$$\nabla_{\mathcal{S}} f(p) \stackrel{\triangle}{=} \nabla f(p) - (\nabla f(p) \cdot \nabla\psi(p))\ \nabla\psi(p)$$

We often use the alternative notation $\nabla_\psi f$ since the definition can be applied to any level set of ψ. Note that we can write $\nabla_\psi f = \mathbf{\Pi}_{\nabla\psi} \nabla f$ where

$$\mathbf{\Pi}_{\nabla\psi} \stackrel{\triangle}{=} \mathbf{I} - \nabla\psi\nabla\psi^T$$

Implicit Hessian

If we compute the second derivative of F we find that $\ddot{F}(0) = \nabla f(p) \cdot \ddot{\gamma}(0) + \mathbf{H}_f[\dot{\gamma}(0), \dot{\gamma}(0)]$. Now, we know that an arc-length parameterized geodesic curve of $\mathcal{S}$ must satisfy the *harmonic maps* differential equation

$$\ddot{\gamma} + \mathbf{H}_\psi(\gamma)[\dot{\gamma}, \dot{\gamma}]\ \nabla\psi(\gamma) = 0$$

We then find that $\ddot{F}(0) = (\mathbf{H}_f(p) - \nabla f(p) \cdot \nabla\psi(p)\ \mathbf{H}_\psi(p))[\dot{\gamma}, \dot{\gamma}]$. Again we have that $\dot{\gamma} \in T_p\mathcal{S}$, and we find the implicit Hessian of f at p to be

$$\mathbf{H}_f^{\mathcal{S}}(p) \stackrel{\triangle}{=} \mathbf{\Pi}_\psi \mathbf{h}_f \mathbf{\Pi}_\psi$$

where

$$\mathbf{h}_f \stackrel{\triangle}{=} \mathbf{H}_f(p) - \nabla f(p) \cdot \nabla\psi(p)\ \mathbf{H}_\psi(p)$$

We will frequently use the alternative notation $\mathbf{H}_f^\psi(p)$.

Implicit Laplacian

From the previous computation it's an easy exercise to compute the implicit Laplacian or Laplace-Beltrami of f since by definition $\Delta_{\mathcal{S}} f = trace\{\mathbf{H}_f^{\mathcal{S}}\}$.

For any pair of symmetric matrices $\mathbf{A}$ and $\mathbf{B}$ one has that $trace\{\mathbf{ABA}\} = \sum_i \sum_j \sum_k a_{ij} a_{ik} b_{jk}$ and $trace\{\mathbf{AB}\} = \sum_i \sum_j a_{ij} b_{ij}$. Now we have that $\mathbf{\Pi}_\psi \mathbf{B} \mathbf{\Pi}_\psi = \mathbf{B} + \nabla\psi\nabla\psi^T\mathbf{B}\nabla\psi\nabla\psi^T - \nabla\psi\nabla\psi^T\mathbf{B} - \mathbf{B}\nabla\psi\nabla\psi^T$. We then obtain

$$
\begin{aligned}
trace\{\mathbf{\Pi}_\psi \mathbf{B} \mathbf{\Pi}_\psi\} &= trace\{\mathbf{B}\} + \sum_i \sum_j \sum_k \psi_{x_i} \psi_{x_j} \psi_{x_i} \psi_{x_k} b_{jk} \\
&- 2 \sum_i \sum_j \psi_{x_i} \psi_{x_j} b_{ij}
\end{aligned}
$$

Recalling that $\psi(\cdot)$ is a *distance function,* so that it satisfies $\|\nabla\psi\| = 1$, we find

$$
\begin{aligned}
trace\{\mathbf{\Pi}_\psi \mathbf{B} \mathbf{\Pi}_\psi\} &= trace\{\mathbf{B}\} - \sum_i \sum_j \psi_{x_i} \psi_{x_j} b_{ij} \\
&= trace\{\mathbf{B}\} - \mathbf{B}[\nabla\psi, \nabla\psi]
\end{aligned}
$$

We conclude the reasoning by taking $\mathbf{B} = \mathbf{h}_f$:

$$
\begin{aligned}
trace\{\mathbf{H}_f^{\mathcal{S}}\} &= trace\{\mathbf{h}_f\} - \mathbf{h}_f[\nabla\psi, \nabla\psi] \\
&= trace\{\mathbf{h}_f\} - \mathbf{H}_f[\nabla\psi, \nabla\psi]
\end{aligned}
$$

since $\mathbf{H}_\psi[\nabla\psi, \nabla\psi] = 0$. Since $trace\{\mathbf{H}_f\} = \Delta f - (\nabla f \cdot \nabla\psi)\Delta\psi$, we find that

$$
\Delta_{\mathcal{S}} f = \Delta f - (\nabla f \cdot \nabla\psi)\Delta\psi - \mathbf{H}_f[\nabla\psi, \nabla\psi]
$$

It's interesting to observe how the expression just found for $\Delta_{\mathcal{S}} f$ coincides with the one obtained by minimizing the *intrinsic Dirichlet integral,*[20]

$$
D(f) \stackrel{\triangle}{=} \frac{1}{2} \int_{I\!R^d} \|\nabla_{\mathcal{S}} f\|^2 \, \delta(\psi) \, dv
$$

as is done in [3]. The authors showed that a smooth function f extremizing $D(f)$ must satisfy

$$
\nabla \cdot (\nabla f - (\nabla f \cdot \nabla\psi) \ \nabla\psi) = 0
$$

We should verify that this definition coincides with ours. This is accomplished as follows:

$$
\begin{aligned}
\nabla \cdot (\nabla f - (\nabla f \cdot \nabla\psi) \ \nabla\psi) &= \Delta f - (\nabla f \cdot \nabla\psi)\Delta\psi - \nabla(\nabla f \cdot \nabla\psi) \cdot \nabla\psi \\
&= \Delta f - (\nabla f \cdot \nabla\psi)\Delta\psi - \mathbf{H}_f[\nabla\psi, \nabla\psi] - \mathbf{H}_\psi[\nabla f, \nabla\psi] \\
&= \Delta f - (\nabla f \cdot \nabla\psi)\Delta\psi - \mathbf{H}_f[\nabla\psi, \nabla\psi] \\
&= \Delta_{\mathcal{S}} f \ \text{(according to our definition)},
\end{aligned}
$$

since $\mathbf{H}_\psi[\nabla\psi, \bullet] = 0$.

[20] As one expects since this is the definition of *harmonic functions.*

Vector Calculus

- **Implicit Jacobian:** With the ideas developed before, we easily find (differentiating $\vec{\Lambda}(t)$) that

$$\mathbf{J}^{\mathcal{S}}_{\vec{\lambda}} \stackrel{\triangle}{=} J_{\vec{\lambda}} \mathbf{\Pi}_{\psi}$$

- **Implicit Divergence:** Using the expression for the intrinsic Jacobian we write

$$\nabla_{\mathcal{S}} \cdot \vec{\lambda} \stackrel{\triangle}{=} trace\left(\mathbf{J}_{\vec{\lambda}} \mathbf{\Pi}_{\psi}\right)$$

and

$$\nabla_{\mathcal{S}} \cdot \vec{\lambda} \stackrel{\triangle}{=} \nabla \cdot \vec{\lambda} - \mathbf{J}_{\vec{\lambda}}[\nabla\psi, \nabla\psi]$$

It is useful to observe that $\nabla_{\mathcal{S}} \cdot \vec{\lambda} = \nabla \cdot \vec{\lambda}$ when $\vec{\lambda}(x) \in T_x\{\psi = 0\}$

References

[1] F. Alouges, "An energy decreasing algorithm for harmonic maps," in J.M. Coron *et al.*, Editors, *Nematics*, Nato ASI Series, Kluwer Academic Publishers, Netherlands, pp. 1-13, 1991.

[2] M. Bertalmio, PhD Dissertation, University of Minnesota (`www.ece.umn.edu/users/~marcelo`), March 2001.

[3] M. Bertalmio, L. T. Cheng, S. Osher, and G. Sapiro, "Variational problems and partial differential equations on implicit surfaces," *Journal of Computational Physics*, **174:2**, pp. 759-780, 2001.

[4] H. Brezis, J. M. Coron, and E. H. Lieb, "Harmonic maps with defects," *Communications in Mathematical Physics* **107**, pp. 649-705, 1986.

[5] V. Caselles, R. Kimmel, G. Sapiro, and C. Sbert, "Minimal surfaces based object segmentation," *IEEE-PAMI*, **19:4**, pp. 394-398, April 1997.

[6] T. Chan and J. Shen, "Variational restoration of non-flat image features: Models and algorithms," *UCLA CAM-TR 99-20*, June 1999.

[7] Y. Chen, M.C. Hong and N. Hungerbuhler, "Heat flow of p-harmonic maps with values into spheres," *Math. Z.* **205**, pp. 25-35, 1994.

[8] L. T. Cheng, "The level set method applied to geometrically based motion, materials science, and image processing (Ph.D. Thesis)," *CAM-UCLA Report* **00-20**, June 2000

[9] R. Cohen, R. M. Hardt, D. Kinderlehrer, S. Y. Lin, and M. Luskin, "Minimum energy configurations for liquid crystals: Computational results," in J. L. Ericksen and D. Kinderlehrer, Editors, *Theory and Applications of Liquid Crystals*, pp. 99-121, IMA Volumes in Mathematics and its Applications, Springer-Verlag, New York, 1987.

[10] J. M. Corcuera and W. S. Kendall, "Riemmanian barycenters and geodesic convexity," preprint, U. of Warwick.

[11] J. M. Coron and R. Gulliver, "Minimizing p-harmonic maps into spheres," *J. Reine Angew. Mathem.* **401**, pp. 82-100, 1989.

[12] W. E and X. P. Wang. “Numerical Methods for the Landau-Lifshitz Equation,” http://www.math.princeton.edu/~weinan/papers/LL1.pdf

[13] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. “Multiresolution analysis of arbitrary meshes,” *Computer Graphics (SIGGRAPH ’95 Proceedings),* pp. 173-182, 1995.

[14] M. Eck and H. Hoppe, “Automatic reconstruction of B-spline surfaces of arbitrary topological type,” *Computer Graphics (SIGGRAPH),* 1996.

[15] J. Eells and L. Lemarie, “A report on harmonic maps,” *Bull. London Math. Soc.* **10:1**, pp. 1-68, 1978.

[16] J. Eells and L. Lemarie, “Another report on harmonic maps,” *Bull. London Math. Soc.* **20:5**, pp. 385-524, 1988.

[17] O. Faugeras, F. Clément, R. Deriche, R. Keriven, T. Papadopoulo, J. Gomes, G. Hermosillo, P. Kornprobst, D. Lingrad, J. Roberts, T. Viéville, F. Devernay, “The inverse EEG and MEG problems: The adjoint state approach I: The continuous case,” *INRIA Research Report* **3673**, June 1999.

[18] S. F. Frisken, R. N. Perry, A. Rockwood, and T. Jones, “Adaptively sampled fields: A general representation of shape for computer graphics,” *Computer Graphics (SIGGRAPH),* New Orleans, July 2000.

[19] *Flujos Toolbox.* November 1999, http://www.iie.edu.uy/investigacion/grupos/gti/flujos/flujos.html.

[20] M. Giaquinta, G. Modica, and J. Soucek, “Variational problems for maps of bounded variation with values in S^1,” *Cal. Var.* **1**, pp. 87-121, 1993.

[21] B. Gustafsson, H.O. Kreiss and J. Oliger, “Time Dependent Problems and Difference Methods” John Wiley & sons inc.

[22] R. Hamilton, *Harmonic maps of manifolds with boundary,* Lecture Notes in Mathematics **471**, Springer-Verlag, Berlin-New York, 1975.

[23] R. M. Hardt, “Singularities of harmonic maps,” *Bulletin of the American Mathematical Society* **34:1**, pp. 15-34, 1997.

[24] T. Kailath. *Linear Systems,* Prentice Hall, 1980.

[25] T. Kanai, H. Suzuki, and F. Kimura, “Three dimensional geometric metamorphosis based on harmonic maps,” *The Visual Computer* **14:4**, pp.166-176, 1998.

[26] R. Kimmel and N. Sochen, “Orientation diffusion,” *Journal of Visual Communication and Image Representation,* to appear.

[27] V. Krishnamurthy and M. Levoy, “Fitting smooth surfaces to dense polygon meshes,” *Computer Graphics,* pp. 313-324, 1996.

[28] J. M. Lee, *Riemannian Manifolds : An Introduction to Curvature*, Springer Verlag, New York, 1987.

[29] F. Mémoli, *Distance Maps on Implicitly Defined Manifolds,* Master Thesis, Universidad de la Republica, Uruguay, May 2001.

[30] F. Mémoli and G. Sapiro, "Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces," *Journal of Computational Physics*, **173:2**, pp. 730-764, November 2001.

[31] F. Mémoli and G. Sapiro "Harmonic brain warping," in preparation.

[32] S. Nishikawa, "On the Neumann problem for the nonlinear parabolic equation of Eells-Sampson and harmonic mappings," *Math. Ann.* **249**, pp. 177-190, 1980.

[33] S. J. Osher and J. A. Sethian, "Fronts propagation with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics* **79**, pp. 12-49, 1988.

[34] A. Pardo and G. Sapiro, "Vector probability diffusion," *IEEE Signal Processing Letters* **8**, pp. 106-109, April 2001.

[35] P. Perona, "Orientation diffusion," *IEEE Trans. Image Processing* **7**, pp. 457-467, 1998.

[36] T. Sakai, *Riemannian Geometry*, AMS Translations of Mathematical Monographs, vol 149.

[37] N. Sochen, R. Kimmel, and R, Malladi, "A general framework for low level vision," *IEEE Trans. Image Processing* **7**, pp. 310-318, 1998.

[38] M. Struwe, "On the evolution of harmonic mappings of Riemannian surfaces," *Comment. Math. Helvetici* **60**, pp. 558-581, 1985.

[39] M. Struwe, *Variational Methods*, Springer Verlag, New York, 1990.

[40] B. Tang, G. Sapiro, and V. Caselles, "Diffusion of general data on non-flat manifolds via harmonic maps theory: The direction diffusion case," *Int. Journal Computer Vision* **36:2**, pp. 149-161, February 2000.

[41] B. Tang, G. Sapiro, and V. Caselles, "Color image enhancement via chromaticity diffusion," *IEEE Trans. Image Processing* **10**, pp. 701-707, May 2001.

[42] J. W. Thomas, "Numerical Partial Differential Equations, Finite Difference Methods". Texts in Applied Mathematics, 22. Springer Verlag, 1995.

[43] A. W. Toga, *Brain Warping*, Academic Press, New York, 1998.

[44] D. Tschumperle and R. Deriche, "Regularization of orthonormal vector sets using coupled PDE's," *IEEE Workshop on Variational and Level-Set Methods*, Vancouver, Canada, July 2001.

[45] G. Yngve and G. Turk, "Creating smooth implicit surfaces from polygonal meshes," *Technical Report GIT-GVU-99-42, Graphics, Visualization, and Usability Center. Georgia Institute of Technology*, 1999.

[46] D. Zhang and M. Hebert, "Harmonic maps and their applications in surface matching," *Proc. CVPR '99*, Colorado, June 1999.

[47] G. Zigelman, R. Kimmel, and N. Kiryati, "Texture mapping using surface flattening via multi-dimensional scaling," *Technion-CIS Technical Report* **2000-01**, 2000.

[48] L. A. Vese and S. J. Osher, "Numerical methods for p-harmonic flows and applications to image processing," *CAM-UCLA Report* **01-22**, August 2001

[49] `www.kitware.com/VTK`

[50] J. Weickert, *Anisotropic Diffusion in Image Processing*, ECMI Series, Teubner-Verlag, Stuttgart, Germany, 1998.

Isosurfaces and Level-Set Surface Models [a]

Ross T. Whitaker

Technical Report UUCS-02-010

[a] Versions of these notes and the accompanying talk appeared in tutorials at IEEE Visualization 2000 and 2001, and ACM SIGGRAPH 2001 and 2002.

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

April 3, 2002

Abstract

This paper is a set of notes that present the basic geometry of isosurfaces and the basic methods for using level sets to model deformable surfaces. It begins with a short introduction to isosurface geometry, including curvature. It continues with a short explanation of the level-set partial differential equations. It also presents some practical details for how to solve these equations using up-wind scheme and sparse calculation methods. This paper presents a series of examples of how level-set surface models are used to solve problems in graphics and vision. Finally, it presents some examples of implementations using *VISPack*, an object oriented, C++ library for doing volume processing and level-set surface modeling.

1 Introduction

1.1 Motivation

These notes address mechanisms for analyzing and processing volumes in a way that deals specifically with *isosurfaces*. The underlying philosophy is to use isosurfaces as a modeling technology that can serve as an alternative to parameterized models for a variety of important applications in visualization and computer graphics. This paper presents the mathematics and numerical techniques for describing the geometry of isosurfaces and manipulating their shapes in prescribed ways. We start with a basic introduction into the notation and fundamental concepts and then presents the geometry of isosurfaces. We describe the method of level sets, i.e., moving isosurfaces, and present the mathematical and numerical methods they entail. This paper concludes with some application examples and describes *VISPACK*, a *C++*, object-oriented library the performs volume processing and level-set modeling.

1.2 Isosurfaces

1.2.1 Modeling Surfaces With Volumes

When considering surface models for graphics and visualization, one is faced with a staggering variety of options including meshes, spline-based patches, constructive solid geometry, implicit blobs, and particle systems. These options can be divided into two basic classes — explicit (parameterized) models and implicit models. With an implicit model, one specifies the model as a *level set* of a scalar function,

$$\begin{array}{cccc} \phi: & U & \mapsto & \mathbb{R} \\ & x,y,z & & k \end{array}, \tag{1}$$

where $U \subset \mathbb{R}^3$ is the domain of the volume (and the *range* of the surface model). Thus, a surface $\mathcal{S}$ is

$$\mathcal{S} = \{\boldsymbol{x} | \phi(\boldsymbol{x}) = k\}\,. \tag{2}$$

The choice of k is arbitrary, and ϕ is sometimes called the *embedding*. Notice that surfaces defined in this way divide U into a clear inside and outside—such surfaces are always closed wherever they do not intersect the boundary of the domain.

Choosing this implicit strategy begs the question of how to represent ϕ. Historically, implicit models are represented using linear combinations of *basis* functions. These basis or

potential functions usually have several degrees of freedom including 3D position, size, and orientation. By combining these functions, one can create complex objects. Typical models might contain several hundred to several thousands of such primitives. This is the strategy behind the "blobby" models proposed by Blinn [1].

While such an implicit modeling strategy offers a variety of new modeling tools, it has some limitations. In particular, the global nature of the potential functions limits ones ability to model *local* surface deformations. Consider a point $\boldsymbol{x} \in \mathcal{S}$ where $\mathcal{S}$ is the level surface associated with a model $\phi = \sum_i \alpha_i$, and α_i is one of the individual potential functions that comprise that model. Suppose one wishes to move the surface at the point $\boldsymbol{x}$ in a way that maintains continuity with the surrounding neighborhood. With multiple, global basis functions one must decide which basis function or combination of basis functions to alter and at the same time control the effects on other parts of the surface. The problem is generally ill posed — there are many ways to adjust the basis functions so that $\boldsymbol{x}$ will move in the desired direction and yet it may be impossible to eliminate the effects of those movements on other disjoint parts of the surface. These problems can be overcome, however they usually entail heuristics that tie the behavior of the surface deformation to the choice of representation [2].

An alternative to using a small number of *global* basis functions is to use a relatively large number of *local* basis functions. This is the principle behind using a volume as an implicit model. A volume is a discrete sampling of the embedding ϕ. It is also an implicit model with a very large number of basis functions, as shown in Figure 1. The total number of basis functions is fixed, as are their positions (grid points) and extent. One can change only the magnitude of each basis function, i.e., each basis function has only one degree of freedom. A typical volume of size $128 \times 128 \times 128$ contains over a million such basis functions. The shape of each basis function is open to interpretation — it depends on how one interpolates the values between the grid points. A trilinear interpolation, for instance, implies a basis function that is a piece-wise cubic polynomial with a value of one at the grid point and zero at neighboring grid points. Another advantage of using volumes as implicit models, is that for the purposes of analysis we can treat the volume as a continuous function whose values can be *set* at each point according to the application. Once the continuous analysis is complete we can map the algorithm into the discrete domain using standard methods of numerical analysis. The sections that follow discuss how to compute the geometry of surfaces that are represented as volumes and how to manipulate the shapes of those surfaces by changing the gray-scale values in the volume.

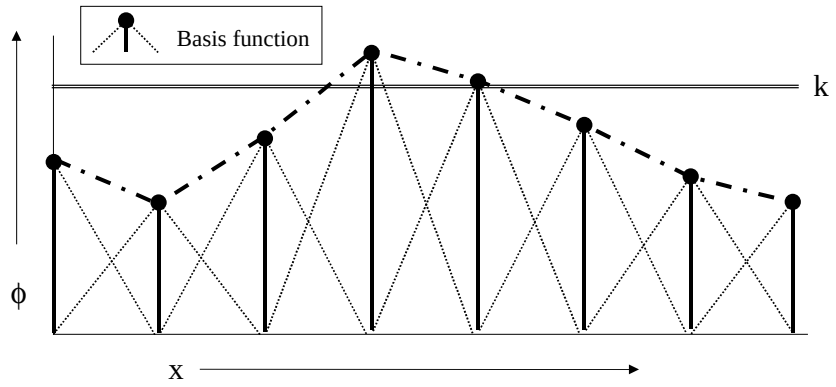


Figure 1: A volume can be considered as an implicit model with a large number of local basis functions.

1.2.2 Isosurface Extraction and Visualization

This paper addresses the question of how to use volumes as surface models. Depending on the application, however, a 3D grid of data (i.e. a volume) may not be a suitable model representation. For instance, if the goal is make measurements of an object or visualize its shape, an explicit model might be necessary. In such cases it is beneficial to convert between volumes and other modeling technologies.

For instance, the literature proposes several methods for scan converting polygonal meshes or solid models [3, 4]. Likewise a variety of methods exist for extracting parametric models of isosurfaces from volumes. The most prevalent method is to locate isosurface crossings along grid lines in a volume (between voxels along the 3 cardinal directions) and then to link these points together to form triangles and meshes. This is the strategy of "marching cubes" [5] and other related approaches. However, extracting a parametric surface is not essential for visualization, and a variety of direct methods [6, 7] are now computationally feasible and arguably superior in quality. These notes do not address the issue of extracting or rendering isosurfaces, but rather studies the geometry of isosurfaces and how to manipulate them directly by changing the grey-scale values in the underlying volume. Thus, we propose volumes as a mechanism for studying and deforming surfaces, regardless of the ultimate form of the output. Their are many ways of rendering or visualizing them and and these techniques are beyond the scope of this discussion.

2 Surface Normals

The surface normal of an isosurface is given by the normalized gradient vector. Typically, we identify a surface normal with a point in the volume domain D. That is

$$\boldsymbol{n}(\boldsymbol{x}) = \frac{\nabla\phi(\boldsymbol{x})}{|\nabla\phi(\boldsymbol{x})|} \quad \text{where} \quad \boldsymbol{x} \in D. \tag{3}$$

The convention regarding the direction of this vector is arbitrary; the negative of the normalized gradient magnitude is also normal to the isosurface. The gradient vector points toward that side of the isosurface which has greater values (i.e. brighter). When rendering, the convention is to use *outward pointing* normals, and the sign of the gradient must be adjusted accordingly. However, for most applications any consistent choice of normal vector will suffice. On a discrete grid, one must also decide how to approximate the gradient vector (i.e., first partial derivatives). In many cases central differences will suffice. However, in the presence of noise, especially when volume rendering, it is sometimes helpful to compute first derivatives using some smoothing filter (e.g., convolution with a Gaussian). When

using the normal vector to solve certain kinds of partial differential equations, it is sometimes necessary to approximate the gradient vector with discrete, one-sided differences, as discussed in successive sections.

Note that a single volume contains families nested isosurfaces, arranged like the layers of an onion. We specific the normal to an isosurface as a function of the position within the volume. That is, $\boldsymbol{n}(\boldsymbol{x})$ is the normal of the (single) isosurface that passes through the point $\boldsymbol{x}$. The k value associated with that isosurface is $\phi(\boldsymbol{x})$.

3 Second-Order Structure

In differential geometric terms, the second-order structure of a surface is characterized by a quadratic patch that shares first- and second-order contact with the surface at a point (i.e., tangent plane and osculating circles). The *principal directions* of the surface are those associated with the quadratic approximation, and the *principal curvatures*, k_1, k_2, are the curvatures in those directions.

The second-structure of the isosurface can be computed from the first- and second-order structure of the embedding, ϕ. All of the isosurface shape information is contained field of normals given by $\boldsymbol{n}(\boldsymbol{x})$. The 3×3 matrix of derivatives of this vector,

$$N = -\left[\boldsymbol{n}_x \;\; \boldsymbol{n}_y \;\; \boldsymbol{n}_z.\right] \tag{4}$$

The projection of this derivative onto the tangent plane of the isosurface gives the shape matrix, β. Let P denote normal projection operator, which is defined as

$$P = \boldsymbol{n} \otimes \boldsymbol{n} = \frac{1}{||\nabla\phi||^2} \begin{pmatrix} \phi_x^2 & \phi_x\phi_y & \phi_x\phi_z \\ \phi_y\phi_x & \phi_y^2 & \phi_y\phi_z \\ \phi_z\phi_x & \phi_z\phi_y & \phi_z^2 \end{pmatrix}. \tag{5}$$

The tangential projection operator is $I - P$, and thus the shape matrix is

$$\beta = NT = TH_\phi T, \tag{6}$$

where H_ϕ is the Hessian of ϕ. The shape matrix β has 3, real, eigenvalues which are

$$e_1 = k_1, e_2 = k_2, e_3 = 0. \tag{7}$$

The corresonding eigenvectors are the principle directions (in the tangent plane) and the normal, respectively.

The *mean curvature* is the mean of the two principal curvatures, which is one half of the trace of β, which is equal to the trace of N:

$$\begin{aligned} H &= \frac{k_1 + k_2}{2} = \frac{1}{2}\mathrm{Tr(N)} \\ &= \frac{\phi_x^2(\phi_{yy} + \phi_{zz}) + \phi_y^2(\phi_{xx} + \phi_{zz}) + \phi_z^2(\phi_{xx} + \phi_{yy}) - 2\phi_x\phi_y\phi_{xy} - 2\phi_x\phi_z\phi_{xz} - 2\phi_y\phi_z\phi_{yz}}{2(\phi_x^2 + \phi_y^2 + \phi_z^2)^{3/2}} \end{aligned} \tag{8}$$

The *Gaussian curvature* is the product of the principal curvatures:

$$\begin{aligned} K &= k_1 k_2 = e_1 e_2 + e_1 e_3 + e_2 e_3 = 2\mathrm{Tr(N)}^2 - \frac{1}{2}||\mathrm{N}|| \\ &= \frac{\begin{array}{c}\phi_z^2(\phi_{xx}\phi_{yy} - \phi_{xy}\phi_{xy}) + \phi_y^2(\phi_{xx}\phi_{zz} - \phi_{xz}\phi_{xz}) + \phi_x^2(\phi_{yy}\phi_{zz} - \phi_{yz}\phi_{yz}) \\ + 2(\phi_x\phi_y(\phi_{xz}\phi_{yz} - \phi_{xy}\phi_{zz}) + \phi_x\phi_z(\phi_{xy}\phi_{yz} - \phi_{xz}\phi_{yy}) + \phi_y\phi_z(\phi_{xy}\phi_{xz} - \phi_{yz}\phi_{xx}))\end{array}}{(\phi_x^2 + \phi_y^2 + \phi_z^2)^2}. \end{aligned} \tag{9}$$

The total curvature, also called the deviation from flatness, D, is the root sum of squares of the two principal curvatures, which is the Euclidean norm of the matrix β.

Notice, these measures exist at every point in U, and at each point they describe the geometry of the particular isosurface that passes through that point. All of these quantities can be computed on a discrete volume using finite differences, as described in successive sections.

4 Deformable Surfaces

This section begins with mathematics for describing surface deformations on parametric models. The result is an evolution equation for a surface. Each of the terms in this evolution equation can be re-expressed in a way that is independent of the parameterization. Finally, the evolution equation for a parametric surface gives rise to an evolution equation (differential equation) on a volume, which encodes the shape of that surface as a level set.

4.1 Surface Deformation

A regular surface $\mathcal{S} \subset \mathbb{R}^3$ is a collection of points in 3D that can be be represented *locally* as a continuous function. In geometric modeling a surface is typically represented as a two-parameter object in a three-dimensional space, i.e., a surface is local a mapping $\boldsymbol{S}$:

$$\boldsymbol{S} : \underset{r}{V} \times \underset{s}{V} \mapsto \underset{x,\, y,\, z,}{\mathbb{R}^3} \tag{10}$$

where $V \times V\mathbb{R}^2$, and the bold notation refers specifically to a parameterized surface (vector-valued function). A deformable surface exhibits some motion over time. Thus $\boldsymbol{S} = \boldsymbol{S}(r, s, t)$, where $t \in \mathbb{R}^+$. We assume second-order-continuous, orientable surfaces; therefore at every point on the surface (and in time) there is surface normal $\boldsymbol{N} = \boldsymbol{N}(r, s, t)$. We use $\mathcal{S}_t$ to refer to the entire set of points on the surface.

Local deformations of $\boldsymbol{S}$ can be described by an evolution equation, i.e., a differential equation on $\boldsymbol{S}$ that incorporates the position of the surface, local and global shape properties, and responses to other forcing functions. That is,

$$\frac{\partial \boldsymbol{S}}{\partial t} = \boldsymbol{G}\left(\boldsymbol{S}, \boldsymbol{S}_r, \boldsymbol{S}_s, \boldsymbol{S}_{rr}, \boldsymbol{S}_{rs}, \boldsymbol{S}_{ss}, \ldots\right), \tag{11}$$

where the subscripts represent partial derivatives with respect to those parameters. The evolution of $\boldsymbol{S}$ can be described by a sum of terms that depends on both the geometry of $\boldsymbol{S}$ and the influence of other functions or data.

There are a variety of differential expressions that can be combined for different applications. For instance, the model could move in response to some directional "forcing" function [8, 9], $\boldsymbol{F} : U \mapsto \mathbb{R}^3$, that is

$$\frac{\partial \boldsymbol{S}}{\partial t} = \boldsymbol{F}(\boldsymbol{S}). \tag{12}$$

Alternatively, the surface could expand and contract with a spatially-varying speed. For instance,

$$\frac{\partial \boldsymbol{S}}{\partial t} = G(\boldsymbol{S})\boldsymbol{N} \tag{13}$$

where $G : \mathbb{R}^3 \mapsto \mathbb{R}$ is a signed speed function. The evolution might also depend on the surface geometry itself. For instance,

$$\frac{\partial \boldsymbol{S}}{\partial t} = \boldsymbol{S}_{rr} + \boldsymbol{S}_{ss} \tag{14}$$

describes a surface that moves in way that is becomes more *smooth* with respect to its own parameterization. This motion can be combined with the motion of Equation 12 to produce a model that is pushed by a forcing function but maintains a certain smoothness in its shape and parameterization. There are myriad terms that depend on both the differential geometry of the surface and outside forces or functions to control the evolution of a surface.

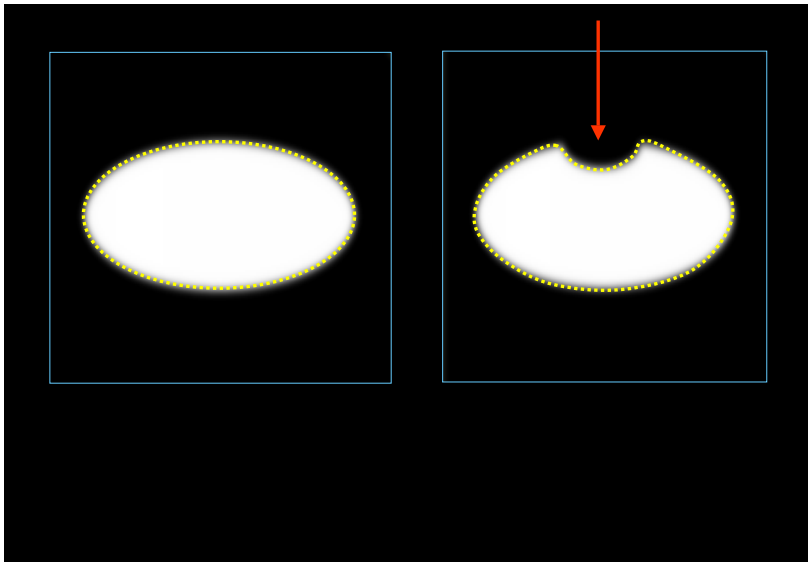


Figure 2: Level-set models represent curves and surfaces implicitly using greyscale images: a) an ellipse is represented as the level set of an image, b) to change the shape we modify the greyscale values of the image.

5 Deformation: The Level Set Approach

The method of level-sets, proposed by Osher and Sethian [10] and described extensively in [11], provides the mathematical and numerical mechanisms for computing surface deformations as time-varying iso-values of ϕ by solving a partial differential equation on the 3D grid. That is, the level-set formulation provides a set of numerical methods that describe how to manipulate the greyscale values in a volume, so that the isosurfaces of ϕ move in a prescribed manner (shown in Figure 2).

We denote the movement of a point on a surface as it deforms as $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$, and we assume that this motion can be expressed in terms of the position of $\boldsymbol{x} \in U$ and the geometry of the surface at that point. In this case, there are generally two options for representing such surface movements implicitly:

Static: A single, static $\phi(\boldsymbol{x})$ contains a family of level sets corresponding to surfaces as different times t. That is,

$$\phi(\boldsymbol{x}(t)) = k(t) \quad \Rightarrow \quad \nabla\phi(\boldsymbol{x}) \cdot \frac{\partial \boldsymbol{x}}{t} = \frac{\mathrm{d}k(t)}{\mathrm{d}t}. \tag{15}$$

To solve this static method requires constructing a ϕ that satisfies equation 15. This is a boundary value problem, which can be solved somewhat efficiently starting with a single surface using the fast marching method of Sethian [12]. This representation has some significant limitations, however, because (by definition) a surface cannot pass back over itself over time, i.e., motions must be strictly monotonic — inward or outward.

Dynamic: The approach is to use a one-parameter *family* of embeddings, i.e., $\phi(\boldsymbol{x}, t)$ changes over time, $\boldsymbol{x}$ remains on the k level set of ϕ as it moves, and k remains constant. The behavior of ϕ is obtained by setting the total derivative of $\phi(\boldsymbol{x}(t), t) = k$ to zero. Thus,

$$\phi(\boldsymbol{x}(t), t) = k \;\; \Rightarrow \;\; \frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}. \tag{16}$$

This approach can accommodate models that move forward and backward and cross back over their own paths (over time). However, to solve this requires solving the initial value problem (using finite forward differences) on $\phi(\boldsymbol{x}, t)$ — a potentially large computational burden. The remainder of this discussion focuses on the dynamic case, because of its superior flexibility.

All surface movements depend on position and geometry, and the level-set geometry is expressed in terms of the differential structure of ϕ. Therefore the dynamic formulation from equation 16 gives a general form of the partial differential equation on ϕ:

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = -\nabla \phi \cdot \boldsymbol{F}(\boldsymbol{x}, \mathrm{D}\phi, \mathrm{D}^2\phi, \ldots), \tag{17}$$

where $D^n\phi$ is the set of order-n derivatives of ϕ evaluated at $\boldsymbol{x}$. Because this relationship applies to every level-set of ϕ, i.e. all values of k, this equation can be applied to all of U, and therefore the movements of *all* the level-set surfaces embedded in ϕ can be calculated from Equation 17.

The level-set representation has a number of practical and theoretical advantages over conventional surface models, especially in the context of deformation and segmentation. First, level-set models are topologically flexible, they can easily represent complicated surface shapes that can, in turn, form holes, split to form multiple objects, or merge with other objects to form a single structure. These models can incorporate many (millions) of degrees of freedom, and therefore they can accommodate complex shapes. Indeed, the shapes formed by the level sets of ϕ are restricted only by the resolution of the sampling. Thus, there is no need to reparameterize the model as it undergoes significant deformations.

Such level-set methods are well documented in the literature [10, 13] for applications such as computational physics [14], image processing [15, 16], computer vision [17, 18], medical image analysis [19, 18], and 3D reconstruction [20, 21]. For instance, in computational

physics level-set methods are a a powerful tool for modeling moving interfaces between different materials (see Osher and Fedkiw [14] for a nice overview of recent results). Examples are water-air and water-oil. In such cases, level-set methods can be used to compute deformations that minimize surface area while preserving volumes for materials that *split and merge* in arbitrary ways. The method can be extended to multiple, non-overlapping objects.

Level-set methods have also been shown to be effective in extracting surface structures from biological and medical data. For instance Malladi *et al.* [18] propose a method in which the level-sets form an expanding or contracting contour which tends to "cling" to interesting features in 2D angiograms. At the same time the contour is also influenced by its own curvature, and therefore remains smooth. Whitaker *et al.* [19, 22] have shown that level sets can be used to simulate conventional deformable surface models, and demonstrated this by extracting skin and tumors from *thick-sliced* (e.g. clinical) MR data, and by reconstructing a fetal face from 3D ultrasound. A variety of authors [23, 24, 16, 25] have presented variations on the method and presented results for 2D and 3D data. Sethian [11] gives several examples of level-set curves and surface for segmenting CT and MR data.

5.1 Deformation Modes

In the case of parametric surfaces, one can choose from a variety of different expressions to construct an evolution equation that is appropriate for a particular application. For each of those parametric expressions, there is a corresponding expression that can be formulated on ϕ, the volume in which the level-set models are embedded. In constructing evolutions on levels sets, there can be no reference to the underlying surface parameterization (terms depending on r and s in Equations 10 through 14). This has two important implications: 1) only those surface movements that are normal to the surface are represented—any other movement is equivalent to a reparameterization 2) all of the derivatives with respect to surface parameters r and s must be expressed in terms of invariant surface properties that can be derived without a parameterization.

Consider the term $\boldsymbol{S}_{rr} + \boldsymbol{S}_{ss}$ from equation 14. If r, s is an orthonormal parameterization, the effect of that term is based purely on surface shape, not on the parameterization, and the expression $\boldsymbol{S}_{rr} + \boldsymbol{S}_{ss}$ is twice the *mean curvature*, H, of the surface. The corresponding level-set formulation is given by Equation 8.

Table 1 shows a list of expressions used in the evolution of parameterized surfaces and their equivalents for level-set representations. Also given are the assumptions about the parameterization that give rise to the level-set expressions.

	Effect	Parametric Evolution	Level-Set Evolution	Parameter Assumptions
1	External force	$\boldsymbol{F}$	$\boldsymbol{F} \cdot \nabla\phi$	None
2	Expansion/ contraction	$G(\boldsymbol{x})\boldsymbol{N}$	$G(\boldsymbol{x})\|\nabla\phi(\boldsymbol{x},t)\|$	None
3	Mean curvature	$S_{rr} + S_{ss}$	$H\|\nabla\phi\|$	Orthonormal
4	Gauss curvature	$S_{rr} \times S_{ss}$	$K\|\nabla\phi\|$	Orthonormal
5	Second order	S_{rr} or S_{ss}	$\left(H \pm \sqrt{H^2 - K}\right)\|\nabla\phi\|$	Principal curvatures

Table 1: A list of evolution terms for parametric models has a corresponding expression on the embedding, ϕ, associated with the level-set models.

6 Numerical Methods

By taking the strategy of embedding surface models in volumes, we have converted equations that describe the movement of surface points to nonlinear, partial differential equations defined on a volume, which is generally a rectilinear grid. The expression $u^n_{i,j,k}$ refers to the nth time step at position i, j, k, which has an associated value in the 3D domain of the continuous volume $\phi(x_i, y_j, z_k)$. The goal is to solve the differential equation consisting of terms from Table 5.1 on the discrete grid $u^n_{i,j,k}$.

The discretization of these equations raises two important issues. First is the availability of accurate, stable numerical schemes for solving these equations. Second is the problem of computational complexity and the fact that we have converted a *surface* problem to a *volume* problem, increasing the dimensionality of the domain over which the evolution equations must be solved.

The level-set terms in Table 1 are combined, based on the needs of the application, to create a partial differential equation on $\phi(\boldsymbol{x}, t)$. The solutions to these equations are computed using finite differences. Along the time axis solutions are obtained using finite *forward* differences, beginning with an initial model (i.e., volume) and stepping sequentially through a series of discrete times steps (which are denoted as superscripts on u). Thus the update equation is:

$$u^{n+1}_{i,j,k} = u^n_{i,j,k} + \Delta t \Delta u^n_{i,j,k}, \tag{18}$$

The term $\Delta u^n_{i,j,k}$ is a discrete approximation to $\partial\phi/\partial t$, which consists of a weighted sum

of terms such as those in Table 5.1. Those terms must, in turn, be approximated using finite differences on the volume grid.

6.1 Up-wind Schemes

The terms in Table 1 fall into two basic categories: the first-order terms (items 1 and 2 in Table 1) and the second-order terms (items 3 through 5). The first-order terms describe a moving wave front with a space-varying velocity (expression 1) or speed (expression 2). Equations of this form cannot be solved with a simple finite forward difference scheme. Such schemes tend to overshoot, and they are unstable. To address this issue Osher and Sethian [26] have proposed an *up-wind* scheme. The up-wind method relies on a one-sided derivative that looks in the up-wind direction of the moving wave front, and thereby avoids the over-shooting associated with finite forward differences.

We denote the type of discrete difference using superscripts on a difference operator, i.e., $\delta^{(+)}$ for forward differences, $\delta^{(-)}$ for backward differences, and δ for central differences. For instance, differences in the x direction on a discrete grid, $u_{i,j,k}$, with domain X and uniform spacing h are defined as

$$\delta_x^{(+)} u_{i,j,k} \triangleq (u_{i+1,j,k} - u_{i,j,k})/h, \tag{19}$$

$$\delta_x^{(-)} u_{i,j,k} \triangleq (u_{i,j,k} - u_{i-1,j,k})/h, \quad \text{and} \tag{20}$$

$$\delta_x u_{i,j,k} \triangleq (u_{i+1,j,k} - u_{i-1,j,k})/(2h), \tag{21}$$

$$\tag{22}$$

where we have left off the time superscript for conciseness. Second-order terms are computed using the *tightest-fitting* central difference operators. For example,

$$\delta_{xx} u_{i,j,k} \triangleq \left(u_{i+1,j,k} + u_{i-1,j,k} - 2u_{i,j,k}\right)/h^2, \tag{23}$$

$$\delta_{zz} u_{i,j,k} \triangleq \left(u_{i,j,k+1} + u_{i,j,k-1} - 2u_{i,j,k}\right)/h^2, \quad \text{and} \tag{24}$$

$$\delta_{xy} u_{i,j,k} \triangleq \delta_x \delta_y u_{i,j,k} \tag{25}$$

The discrete approximation to the first-order terms of in Table 5.1 are computed using the up-wind proposed by Osher and Sethian [10]. This strategy avoids overshooting by approximating the gradient of ϕ using a one-sided differences in the direction that is up-wind of the moving level-set thereby ensuring that no *new* contours are created in the process of updating $u_{i,j,k}^n$ (as depicted in Figure 3). The scheme is separable along each axis (i.e., x, y, and z).

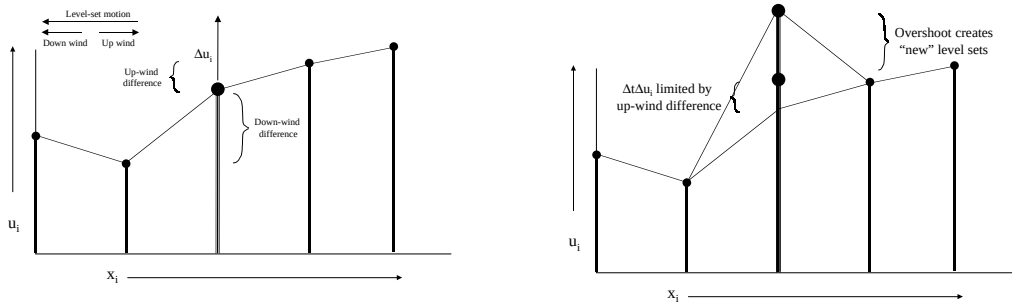


Figure 3: The up-wind numerical scheme uses one-sided derivatives to prevent overshooting and the creation of new level sets.

Consider Term 1 in Table 5.1. If we use superscripts to denote the vector components, i.e.,

$$\boldsymbol{F}(x,y,z) = (F^{(x)}(x,y,z), F^{(y)}(x,y,z), F^{(z)}(x,y,z)), \tag{26}$$

the up-wind calculation for a grid point $u^n_{i,j,k}$ is

$$\boldsymbol{F}(x_i,y_i,z_i)\cdot\nabla\phi(x_i,y_j,z_k,t) \approx \sum_{q\in\{x,y,z\}} F^{(q)}(x_i,y_i,z_i) \begin{cases} \delta_q^+ u^n_{i,j,k} & F^{(q)}(x_i,y_i,z_i) > 0 \\ \delta_q^i u^n_{i,j,k} & F^{(q)}(x_i,y_i,z_i) < 0 \end{cases} \tag{27}$$

The time steps are limited—the fastest moving wave front can move only one grid unit per iteration. That is

$$\Delta t_{\boldsymbol{F}} \leq \frac{1}{\sum_{q\in\{x,y,z\}} \sup_{i,j,k\in X}\{|\nabla F^{(q)}(x_i,y_j,z_k)|\}}. \tag{28}$$

For Term 2 in Table 5.1 the direction of the moving surface depends on the normal, and therefore the same up-wind strategy is applied in a slightly different form.

$$\begin{aligned} &G(x_i,y_j,z_k)|\nabla\phi(x_i,y_j,z_k,t)| \approx \\ &\sum_{q\in\{x,y,z\}} G(x_i,y_i,z_i) \begin{cases} \max^2(\delta_q^+ u^n_{i,j,k},0) + \min^2(\delta_q^- u^n_{i,j,k},0) & G(x_i,y_i,z_i) > 0 \\ \min^2(\delta_q^+ u^n_{i,j,k},0) + \max^2(\delta_q^- u^n_{i,j,k},0) & G(q)(x_i,y_i,z_i) < 0 \end{cases} \end{aligned} \tag{29}$$

The time steps are, again, limited by the fastest moving wave front:

$$\Delta t_G \leq \frac{1}{3\sup_{i,j,k\in X}\{|\nabla G(x_i,y_j,z_k)|\}} \tag{30}$$

To compute approximation the update to the second-order terms in Table 5.1 requires only central differences . Thus, the mean curvature is approximated as:

$$\begin{aligned} H^n_{i,j,k} &= \frac{1}{2}\left(\left(\delta_x u^n_{i,j,k}\right)^2 + \left(\delta_y u^n_{i,j,k}\right)^2 + \left(\delta_z u^n_{i,j,k}\right)^2\right)^{-1} \Big[\left(\left(\delta_y u^n_{i,j,k}\right)^2 + \left(\delta_z u^n_{i,j,k}\right)^2\right)\delta_{xx}u^n_{i,j,k} \\ &\quad + \left(\left(\delta_z u^n_{i,j,k}\right)^2 + \left(\delta_x u^n_{i,j,k}\right)^2\right)\delta_{yy}u^n_{i,j,k} + \left(\left(\delta_x u^n_{i,j,k}\right)^2 + \left(\delta_y u^n_{i,j,k}\right)^2\right)\delta_{zz}u^n_{i,j,k} \\ &\quad - 2\delta_x u^n_{i,j,k}\delta_y u^n_{i,j,k}\delta_{xy}u^n_{i,j,k} - 2\delta_y u^n_{i,j,k}\delta_z u^n_{i,j,k}\delta_{yz}u^n_{i,j,k} - 2\delta_z u^n_{i,j,k}\delta_x u^n_{i,j,k}\delta_{zx}u^n_{i,j,k}\Big] \end{aligned} \tag{31}$$

Such curvature terms can be computing by using a combination of forward and backward differences as described in [27]. In some cases this is advantageous—but the details are beyond the scope of this paper.

The time steps are limited, for stability, to

$$\Delta t_H \leq \frac{1}{6}. \tag{32}$$

When combining terms, the maximum time steps for each terms is scaled by one over the weighting coefficient for that term.

6.2 Narrow-Band Methods

If one is interested in only *a single level set*, the formulation described previously is not efficient. This is because solutions are usually computed over the entire domain of ϕ. The solutions, $\phi(x, y, z, t)$ describe the evolution of an embedded family of contours. While this dense family of solutions might be advantageous for certain applications, there are other applications that require only a single surface model. In such applications the calculation of solutions over a dense field is an unnecessary computational burden, and the presence of contour families can be a nuisance because further processing might be required to extract the level set that is of interest.

Fortunately, the evolution of a single level set, $\phi(\boldsymbol{x}, t) = k$, is not affected by the choice of embedding. The evolution of the level sets is such that they evolve independently (to within the error introduced by the discrete grid). Furthermore, the evolution of ϕ is important only in the vicinity of that level set. Thus, one should perform calculations for the evolution of ϕ only in a neighborhood of the surface $\mathcal{S} = \{\boldsymbol{x} | \phi(\boldsymbol{x}) = k\}$. In the discrete setting, there is a particular subset of grid points whose values control a particular level set (see Figure 4). Of course, as the surface moves, that subset of grid points must change to account for its new position.

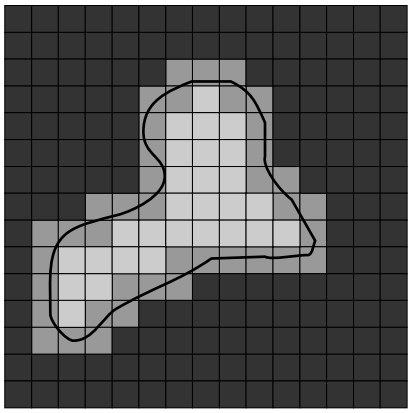


Figure 4: A level curve of a 2D scalar field passes through a finite set of cells. Only those grid points nearest to the level curve are relevant to the evolution of that curve.

Adalsteinson and Sethian [28] propose a *narrow-band* approach which follows this line of reasoning. The narrow-band technique constructs an embedding of the evolving curve or surface via a signed distance transform. The distance transform is truncated, i.e, computed over a finite width of only m points that lie within a specified distance to the level set. The remaining points are set to constant values to indicate that they do not lie within the narrow band, or *tube* as they call it. The evolution of the surface (they demonstrate it for curves in the plane) is computed by calculating the evolution of u only on the set of grid points that are within a fixed distance to the initial level set, i.e. within the narrow band. When the evolving level set approaches the edge of the band (see Figure 5), they calculate a new distance transform and a new embedding, and they repeat the process. This algorithm relies on the fact that the embedding is not a critical aspect of the evolution of the level set. That is, the embedding can be transformed or recomputed at any point in time, so long as such a transformation does not change the position of the kth level set, and the evolution will be unaffected by this change in the embedding.

Despite the improvements in computation time, the narrow-band approach is not optimal for several reasons. First it requires a band of significant width ($m = 12$ in the examples of [28]) where one would like to have a band that is only as wide as necessary to calculate the derivatives of u near the level set (e.g. $m = 2$). The wider band is necessary because the narrow-band algorithm trades off two competing computational costs. One is the cost of stopping the evolution and computing the position of the curve and distance transform (to sub-cell accuracy) and determining the domain of the band. The other is the cost of computing the evolution process over the entire band. The narrow-band method also requires additional techniques, such as smoothing, to maintain the stability at the boundaries of the band, where some grid points are undergoing the evolution and nearby neighbors are static.

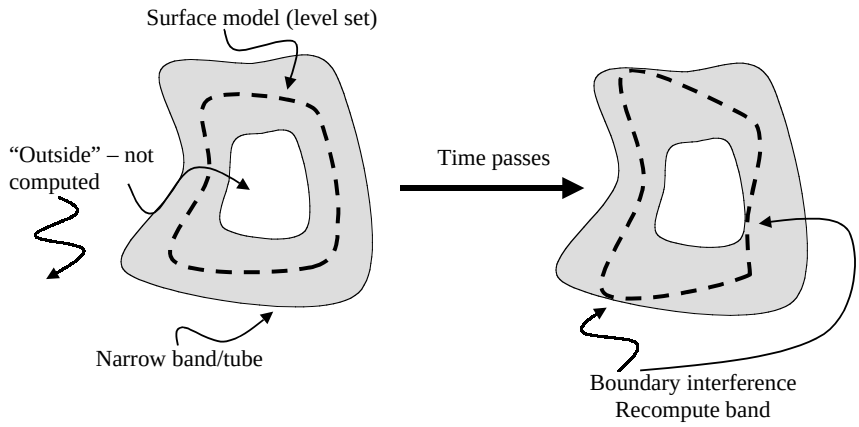


Figure 5: The narrow band scheme limits computation to the vicinity of the specific level set. As the level-set moves near the edge of the band the process is stopped and the band recomputed.

6.3 The Sparse-Field Method

The basic premise of the narrow band algorithm is that computing the distance transform is so costly that it cannot be done at every iteration of the evolution process. The strategy proposed here is to use an approximation to the distance transform that makes it feasible to recompute the neighborhood of the level-set model at each time step. Computation of the evolution equation is computed on a band of grid points that is only on point wide. The embedding is extended from the active points to a neighborhood around those points that is precisely the width needed at each time. This extension is done via a fast distance transform approximation.

This approach has several advantages. First, the algorithm does precisely the number of calculations needed to compute the next position of the level curve. It does not require explicitly recalculating the positions of level sets and their distance transforms. Because the number of points being computed is so small, it is feasible to use a linked-list to keep track of them. Thus, at each iteration the algorithm visits only those points adjacent to the k-level curve. For large 3D data sets, the very process of incrementing a counter and checking the status of all of the grid points is prohibitive.

The *sparse-field* algorithm is analogous to a locomotive engine that lays down tracks before it and picks them up from behind. In this way the number of computations increases with the surface area of the model rather than the resolution of the embedding. Also, the sparse-field approach identifies a single level set with a specific set of points whose values control the position of that level set. This allows one to compute external forces to an accuracy that is better than the grid spacing of the model, resulting in a modeling system that is more accurate for various kinds of "model fitting" applications.

The sparse-field algorithm takes advantage of the fact that a k-level surface, S, of a discrete image u (of any dimension) has a set of cells through which it passes, as shown in Figure 4. The set of grid points adjacent to the level set is called the *active set*, and the individual elements of this set are called *active points.* As a first-order approximation, the distance of the level set from the center of any active point is proportional to the value of u divided the gradient magnitude at that point. Because all of the derivatives (up to second order) in this approach are computed using nearest neighbor differences, only the active points and their neighbors are relevant to the evolution of the level-set at any particular time in the evolution process. The strategy is to compute the evolution given by equation 17 on the active set and then update neighborhood around the active set using a fast distance transform. Because active points must be adjacent to the level-set model, their positions lie within a fixed distance to the model. Therefore the values of u for locations in the active set must lie within a certain range. When active-point values move out of this *active range*

they are no longer adjacent to the model. They must be removed from the set and other grid points, those whose values are moving into the active range, must be added to take their place. The precise ordering and execution of these operations is important to the operation of the algorithm.

The values of the points in the active set can be updated using the up-wind scheme for first-order terms and central differences for the mean-curvature flow, as described in the previous sections. In order to maintain stability, one must update the neighborhoods of active grid points in a way that allows grid points to enter and leave the active set without those changes in status affecting their values. Grid points should be removed from the active set when they are no longer the nearest grid point to the zero crossing. If we assume that the embedding u is a discrete approximation to the distance transform of the model, then the distance of a particular grid point, $x_m = (i, j, k)$, to the level set is given by the value of u at that grid point. If the distance between grid points is defined to be unity, then we should remove a point from the active set when the value of u at that point no longer lies in the interval $[-\frac{1}{2}, \frac{1}{2}]$ (see Figure 6). If the neighbors of that point maintain their distance of 1, then those neighbors will move into the active range just x_m is ready to be removed.

There are two operations that are significant to the evolution of the active set. First, the values of u at active points change from one iteration to the next. Second, as the values of active points pass out of the active range they are removed from the active set and other, neighboring grid points are added to the active set to take their place. In [21] the author gives some formal definitions of active sets and the operations that affect them, which show that active sets will always form a boundary between positive and negative regions in the image, even as control of the level set passes from one set off active points to another.

Because grid points that are near the active set are kept at a fixed value difference from the active points, active points serve to control the behavior of non-active grid points to which they are adjacent. The neighborhoods of the active set are defined in *layers*, $L_{+1}, \ldots L_{+N}$ and $L_{-1}, \ldots L_{-N}$, where the i indicates the distance (city block distance) from the nearest active grid point, and negative numbers are used for the outside layers. For notational convenience the active set is denoted L_0.

The number of layers should coincide with the size of the footprint or neighborhood used to calculate derivatives. In this way, the inside and outside grid points undergo no changes in their values that affect or distort the evolution of the zero set. Most of the level-set work relies on surface normals and curvature, which require only second-order derivatives of ϕ. Second-order derivatives are calculated using a $3 \times 3 \times 3$ kernel (city-block distance 2 to the corners). Therefore only five layers are necessary (2 inside layers, 2 outside layers, and the active set). These layers are denoted L_1, L_2, L_{-1}, L_{-2}, and L_0.

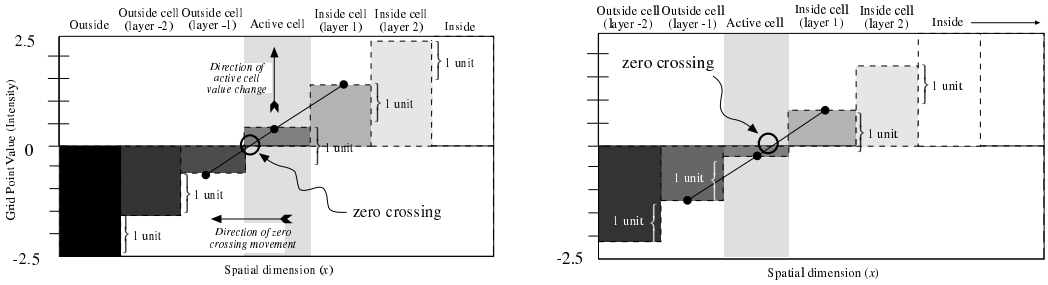


Figure 6: The status of grid points and their values at two different points in time show that as the zero crossing moves, *activity* is passed one grid point to another.

The active set has grid point values in the range $[-\frac{1}{2}, \frac{1}{2}]$. The values of the grid points in each neighborhood layer are kept 1 unit from the next layer closest to the active set (as in Figure 6). Thus the values of layer L_i fall in the interval $[i - \frac{1}{2}, i + \frac{1}{2}]$. For $2N + 1$ layers, the values of the grid points that are totally inside and outside are $N + \frac{1}{2}$ and $-N - \frac{1}{2}$, respectively. The procedure for updating the image and the active set based on surface movements is as follows:

1. For each active grid point, $x_m = (i, j, k)$, do the following:
   (a) Calculate the local geometry of the level set.
   (b) Compute the net change of u_{x_m}, based on the internal and external forces, using some stable (e.g., up-wind) numerical scheme where necessary.
2. For each active grid point x_j add the change to the grid point value and decide if the new value $u_{x_m}^{n+1}$ falls outside the $[-\frac{1}{2}, \frac{1}{2}]$ interval. If so, put x_m on lists of grid points that are changing status, called the *status list*; S_1 or S_{-1}, for $u_{x_m}^{n+1} > 1$ or $u_{x_m}^{n+1} < -1$, respectively.
3. Visit the grid points in the layers L_i in the order $i = \pm 1, \ldots \pm N$, and update the grid point values based on the values (by adding or subtracting one unit) of the next inner layer, $L_{i \mp 1}$. If more than one $L_{i \mp 1}$ neighbor exists then use the neighbor that indicates a level curve closest to that grid point, i.e., use the maximum for the outside layers and minimum for the inside layers. If a grid point in layer L_i has no $L_{i \mp 1}$ neighbors, then it gets demoted to $L_{i \pm 1}$, the next level away from the active set.
4. For each status list $S_{\pm 1}, S_{\pm 2}, \ldots, S_{\pm N}$ do the following:

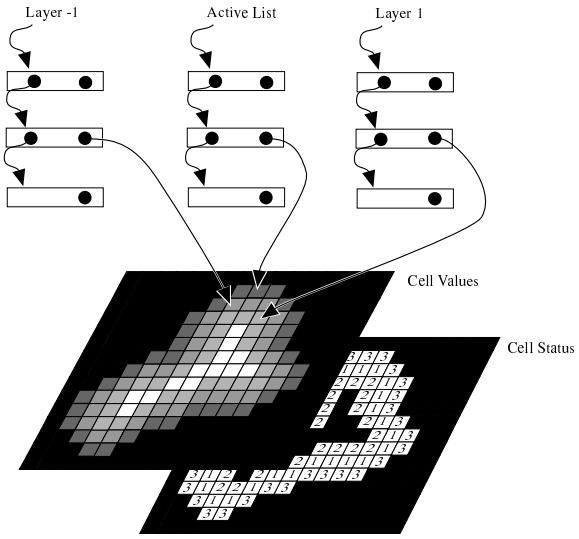


Figure 7: Linked-list data structures provide efficient access to those grid points with values and status that must be updated.

(a) For each element x_j on the status list S_i, remove x_j from the list $L_{i\mp1}$, and add it to the L_i list, or, in the case of $i = \pm(N+1)$, remove it from all lists.

(b) Add all $L_{i\mp1}$ neighbors to the $S_{i\pm1}$ list.

This algorithm can be implemented efficiently using linked-list data structures combined with arrays to store the values of the grid points and their states as shown in Figure 7. This requires only those grid points whose values are changing, the active points and their neighbors, to be visited at each time step. The computation time grows as m^{n-1}, where m is the number of grid points along one dimension of u (sometimes called the resolution of the discrete sampling). Computation time for dense-field approach increases as m^n. The m^{n-1} growth in computation time for the sparse-field models is consistent with conventional (parameterized) models, for which computation times increase with the resolution of the domain, rather than the range.

Another important aspect of the performance of the sparse-field algorithm is the larger time steps that are possible. The time steps are limited by the speed of the "fastest" moving level curve, i.e., the maximum of the force function. Because the sparse-field method calculates the movement of level sets over a subset of the image, time steps are bounded from below by those of the dense-field case, i.e.,

$$\sup_{x \in \mathcal{A} \subset X} (g(x)) \leq \sup_{x \in X} (g(x)), \tag{33}$$

where $g(x)$ is the space varying speed function and $\mathcal{A}$ is the active set.

Results from previous work [21] have demonstrated several important aspects of the sparse-field algorithm. First, the manipulations of the active set and surrounding layers allow the active set to "track" the deformable surface as it moves. The active set always divides the inside and outside of the objects it describes (i.e., it stays closed). Empirical results show significant increases in performance relative to both the computation of full domain and the narrow-band method, as proposed in the literature. Empirical results also show that the sparse-field method is about as accurate as both the full, discrete solution, and the narrow-band method. Finally, because the method positions level sets to sub-voxel accuracy it avoids aliasing problems and is more accurate then these other methods when it comes to *fitting* level-set models to other surfaces. This sub-voxel accuracy is important aspect of the implementation, and will significantly impact the quality of the results for the applications that follow.

7 Applications

This section describes several examples of how level-set surface models can be used to address problems in graphics, visualization, and computer vision. These examples are a small selection of those available in the literature. All of these examples where implemented using the sparse-field algorithm and the VISPack library, which is described in the section that follows.

7.1 Surface Morphing

This section summarizes the work of [29], which describes the use of level-set surface models to perform 3D shape metamorphosis. The *morphing* of 3D surfaces is the process of constructing a series of 3D models that constitute a smooth transition from one shape to another (i.e., a homotopy). Such a capability is interesting for creating animations and as a tool for geometric modeling. There is not yet a single, general method for generating such transitional shapes. However, there are several desirable aspects of morphing algorithms that allow us to compare the adequacy of different approaches to surface morphing. Several desirable properties of 3D surface morphing are:

1. The transition process should begin with an *initial* surface and end with a specified *target* surface.

2. The morphing algorithm should apply to a wide range of shapes and topologies.

3. Intermediate surfaces should undergo continuous 3D transitions (rather than continuity only in the image space).

4. A 3D morphing algorithm should incorporate user input easily but should degrade gracefully without it.

5. Transitional shapes should depend only on the surface geometry of the two input shapes and user input.

These requirements are not exhaustive, but they capture many of the practical aspects of 3D morphing.

In this section we show how level-set models provide an algorithm for 3D morphing which meets most of these criteria and compare favorably with existing algorithms. Furthermore, this algorithm is a natural extension of the mathematical principles discussed in previous sections. The strategy is to allow a free-form deformation of one surface (called the *initial* surface) using the signed distance transform of a second surface (the *target* surface). This free-form deformation is combined with an underlying coordinate transformation that gives either a rough global alignment of the two surfaces, or one-to-one relationships between a finite set of landmarks on both the initial and target surfaces. The coordinate transformation can be computed automatically or using user input (as in [30]).

Much of the previous 3D morphing work has focused on morphing parametric models [31, 32] and applies to only very limited classes of shapes and topologies. Several authors have described volumetric techniques. Hughes [33] demonstrates how volumes can provide topological flexibility in surface morphing. Lerios et al. [30] followed up with a volume-based scheme which incorporates user input via underlying coordinate transformations (a known generalization the image warping technique that is often used in image morphing). Neither of these approaches have dealt with the deeper issue of deforming the level sets of a volume, but rather rely on the properties of the embedding. Payne and Toga [34] as well as Cohen-Or *et al.* [35] fix the embedding problem by using a signed distance transform to create volumes from surfaces. However, interpolating distance transforms can introduce artifacts that violate the previously stated properties, and both of these methods use a discrete distance transform which introduces volume aliasing.

7.1.1 Free-Form Deformations

The distance transform gives the nearest Euclidean distance to a set of points, curve, or surface. For closed surfaces in 3D, the signed distance transform gives a positive distance for points inside and negative for points outside (one can also choose the opposite sign convention).

If two connected shapes overlap then the initial surface can expand or contract using the distance transform of the target. The steady state of such a deformation process is a shape consisting of the zero set of the distance transform of the target. That is, the initial object becomes the target. This is the basis of the proposed 3D morphing algorithm.

Let $D(\boldsymbol{x})$ be the signed distance transform of the target surface, B, and let A be the initial surface. The evolution process which takes a model S from A to B is defined by

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{N}\ D(\boldsymbol{x}), \tag{34}$$

where $\boldsymbol{x}(t) \in \mathcal{S}_t$ and $\mathcal{S}_{t=0} = A$. The free-form deformations can be combined with an underlying coordinate transformation. The strategy is to use a coordinate transformation (for instance a translation and rotation) to position the two surfaces near each other. These transformations can capture gross similarities in shape as well as user input. A coordinate transformation is given by

$$\boldsymbol{x}' = T(\boldsymbol{x}, \alpha), \tag{35}$$

where $0 \leq \alpha \leq 1$ parameterizes a continuous family of these transformations that begins with identity, i.e. $\boldsymbol{x} = T(\boldsymbol{x}, 0)$. The evolution equation for a parametric surface is

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{N}\ D(T(\boldsymbol{x}, 1)), \tag{36}$$

and the corresponding level-set equation is

$$\frac{\partial \Phi(\boldsymbol{x}, t)}{\partial t} = |\nabla \Phi(\boldsymbol{x}, t)|\ D(T(\boldsymbol{x}, 1)). \tag{37}$$

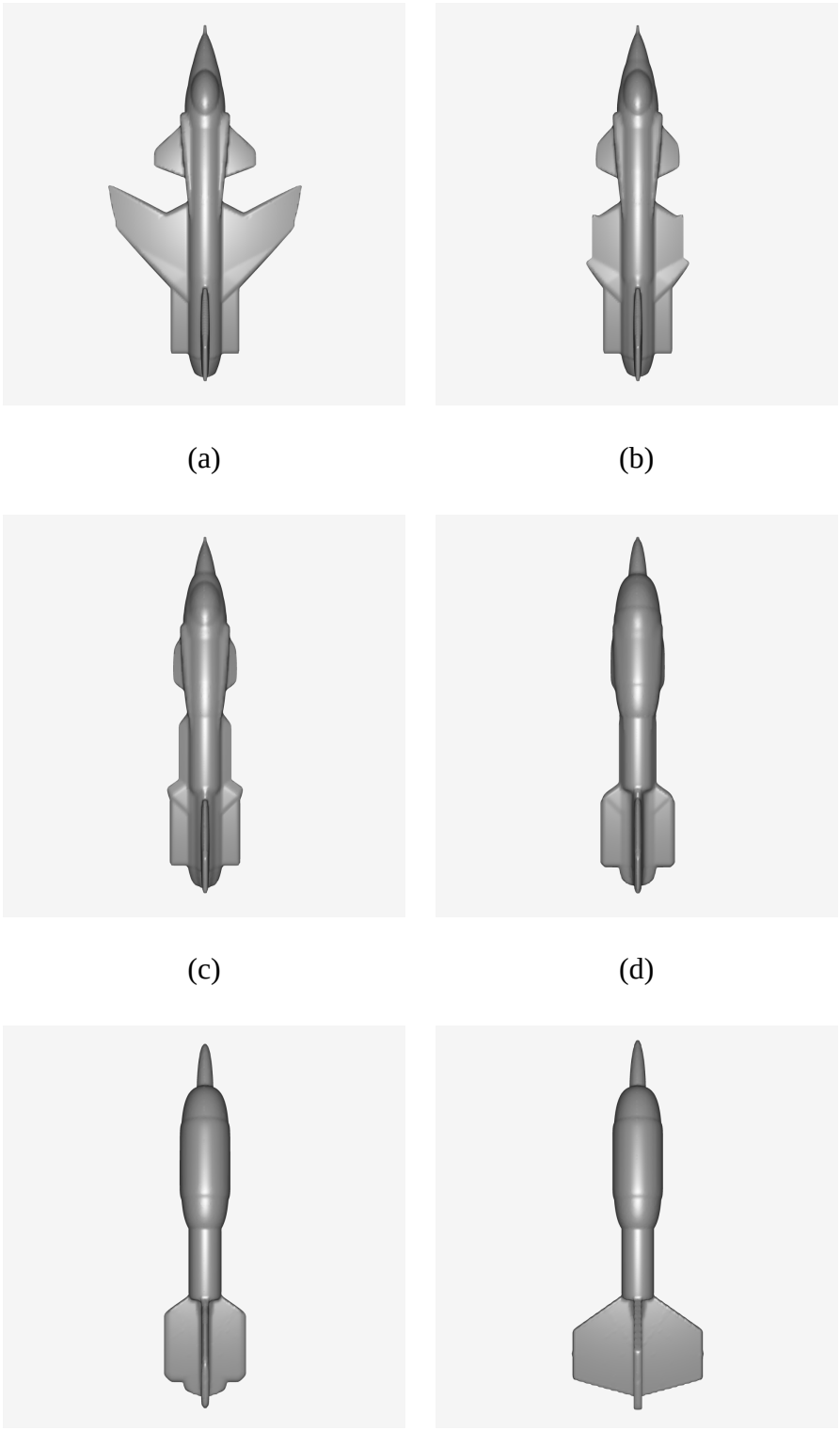
This process produces a series of transition shapes (parameterized by t). The coordinate transformation can be a global rotation, translation, or scaling, or it might be a *warping of the underlying 3D space* as was used by [30]. Incorporating user input is important for any surface morphing technique, because in many cases finding the best set of transition surfaces depends on context. Only users can apply semantic considerations to the transformation of one object to another. However, this underlying coordinate transformation

Figure 8: A 3D model of a jet that was built using Clockworks, a CSG modeling system.

can, in general, achieve only some finite similarity between the "warped" initial model and the target, and even this may require a great deal of user input. In the event that a user is not able or willing to define every important correspondence between two objects, some other method must "fill in" the gaps remaining between the initial and target surface. In [30] they propose alpha blending to achieve that smooth transition—really just a fading from one surface to the other. We are proposing the use of the free-form deformations, implemented with level-set models, to achieve a continuous transition between the shapes that result from the underlying coordinate transformation. We have also experimented with ways of automatically orienting and scaling objects, using 3D moments, in order to achieve a significant correspondence between two objects.

Figure 8 shows a 3D model of a jet that was built using Clockworks [36], a CSG modeling system. Lerios et al. [30] demonstrate the transition of a jet to a dart, which was accomplished using 37 user-defined correspondences, roughly a hundred user-defined parameters. Figure 9 shows the use of level-set models to construct a set of transition surfaces between a jet and a dart. The triangle mesh is extracted from the volume using the method of marching cubes [5]. These results are obtained without any user input. Distance transforms on the CSG models are computed near the level surface using an analytical description and extended into the volume using a level-set method [37].

The application in this section shows how level-set models moving according to the first-order term given in expression 2 in Table 1 can "fit" other objects by moving with a speed that depends on the signed distance transform of the target object. The application in the



(a) (b)

(c) (d)

(e) (f)

Figure 9: The deformation of the jet to a dart using a level-set model moving with a speed defined by the signed distance transform of the target object.

next section relies on expression 5 of Table 1, a second-order flow that depends on the principal curvatures of the surface itself.

7.2 Filleting and Blending Solid Objects

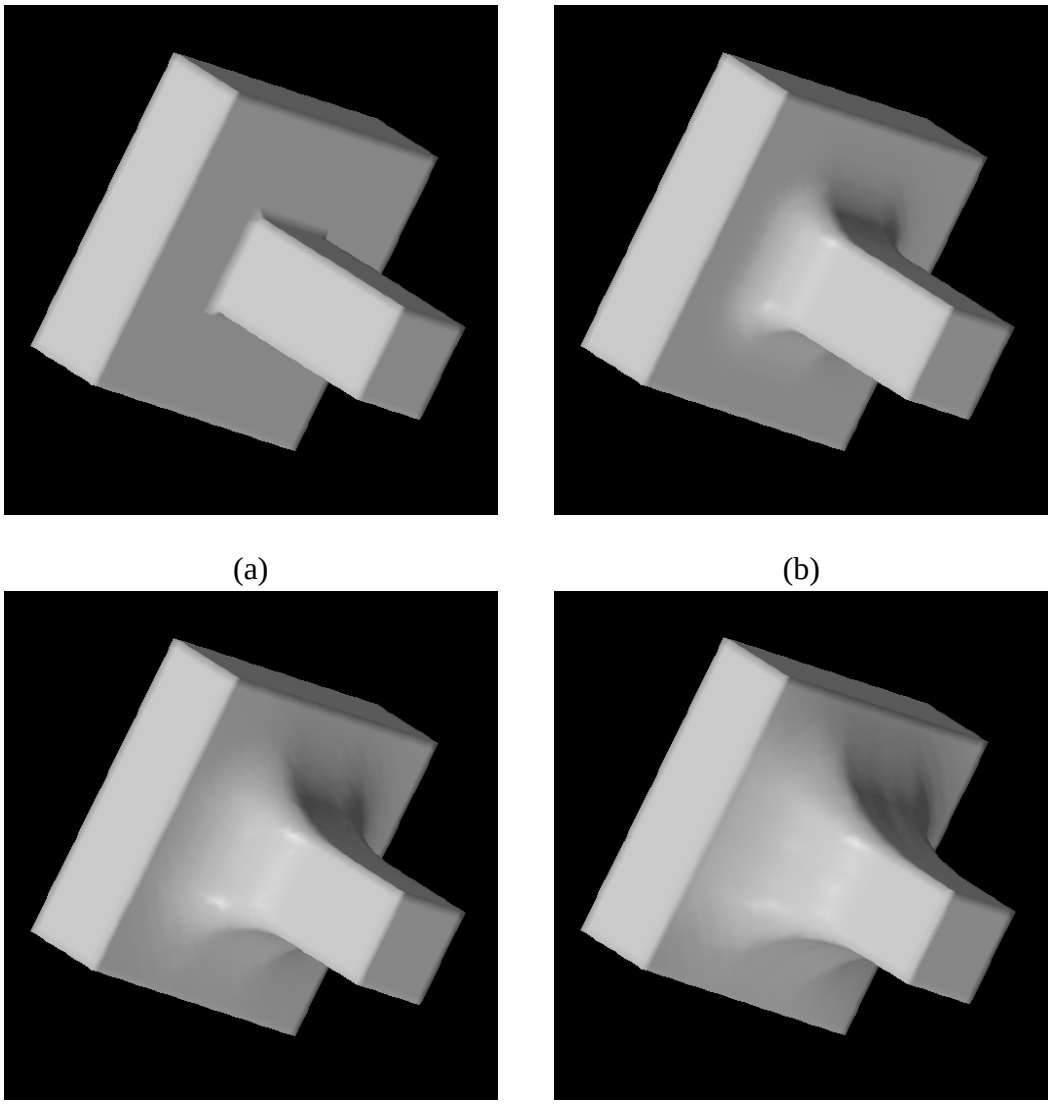
The construction of blending surfaces is an important tool in solid modeling. Geometric solid primitives and their intersections often produce sharp corners or creases that are often not consistent with the real-world objects that they are intended to represent. This section shows how blending can be described as a deformation process, where surfaces move under a geometric flow that can add or remove material based on local curvature information. The result is a method for solid object blending that does not depend on any particular model representation. Thus this method is not restricted to a specific class of shapes or topologies. Additionally, the results are invariant; they do not depend on arbitrary choices of coordinate systems or bases. The only requirement is that the blended objects must be closed surfaces with some known inside-outside function.

Surface blending techniques are typically tied very closely to the choice of geometric primitives. For instance, Middleditch and Sears [38] propose a set-theoretic method for blending solids which relies on low-order algebraic primitives. A fillet at the joint of two tori requires the solution of a degree 32 polynomial. Bloomenthal and Shoemake [39] propose a modeling system based on convolutions, which relies on a skeletonized representation of objects. In general the use of convolution to achieve deformations on implicit shapes results in shapes that reflect both the shape of the model and the embedding, Φ.

The blending method proposed in this section implements an interative smoothing scheme that smooths only along the level set; the final result is independent of the embedding. Consider the case of fillets. We propose that a fillet can be constructed from a process of "filling in" material in places of high curvature. The curvature of a level-set model can be calculated from the embedding, and the deformation of the level set is well defined by the curvature terms in Table 1.

The strategy is to construct a curvature term, k_p, that consists of only positive curvatures. [1] The principal curvatures of the level sets of Φ are functions of Φ and its derivatives. For a specific Φ the principal curvatures are functions of 3-space $k_1(\boldsymbol{x})$ and $k_2(\boldsymbol{x})$. For *adding* material the joint between two objects, we consider only the positive curvature components,

[1] The sign of curvature is defined by the direction of the normals— in this work normals point into the volume enclosed by the object.



(a) (b)

(c) (d)

Figure 10: Two rectangular solid models are joined by a volumetric fillet that is created from a positive curvature flow.

i.e.,

$$\frac{\partial \Phi}{\partial t} = |\nabla \Phi| k_p = |\nabla \Phi| k_1^+ + |\nabla \Phi| k_2^+, \tag{38}$$

where k^+ consists of only the positive parts of k and is defined as zero elsewhere. Because the use of separate curvature terms can cause over-shooting, the up-wind scheme (treating k_p as a space-varying velocity in the normal direction) is used for this evolution.

Figure 10 shows how the positive-curvature flow can be used to construct fillets. No knowledge of the underlying models is necessary. The fillets grow larger as more time passes. The physical extent or position of the fillet can be controlled by either specifying a region of action or by placing a small blob of deformable material in the joint that requires a fillet. Figure 11 shows how such a blending capability can be useful in animation. In this case a pair of superquadrics undergo a rigid transformation that controls their relative positions. Level-set models with a positive-curvature flow are used to create a smooth joint between these two primitives. Notice that the positive curvature method does not suffer from the growth or expansion artifacts that are often associated with distance-based blending methods [40].

Thus, a second-order flow can create smooth blends between objects in a way that does not require specific knowledge of the shapes or topologies of the object involved. The application in the next section, 3D scene reconstruction, shows how a combination of first-order and second-order terms from Table 1 are combined to create technique that fits models to data while maintaining certain smoothness constraints and thereby offsetting the effects of noise.

7.3 3D Reconstruction from Multiple Range Maps

Level-set models are useful for problems related to 3D reconstruction. Previous work has presented level-set results derived from noisy 3D data such as MRI [19] and ultrasound [41]. In [42] we have shown how the reconstruction of objects from multiple range maps can be formulated as a problem of finding the surface that optimizes the posterior probability given a set of measurements (noisy range maps) and some information about the a-priori probability of different kinds of surfaces. That optimization problem can be expressed as a volume integral which can be solved with level-set models. This section presents the mathematical expressions that result from those formulations and presents some new results: the reconstruction of entire scenes by fitting level-set models to the data from a scanning LADAR (laser ranging and detection) system.

A *range map* is a collection of range measurements taken along different directions (lines

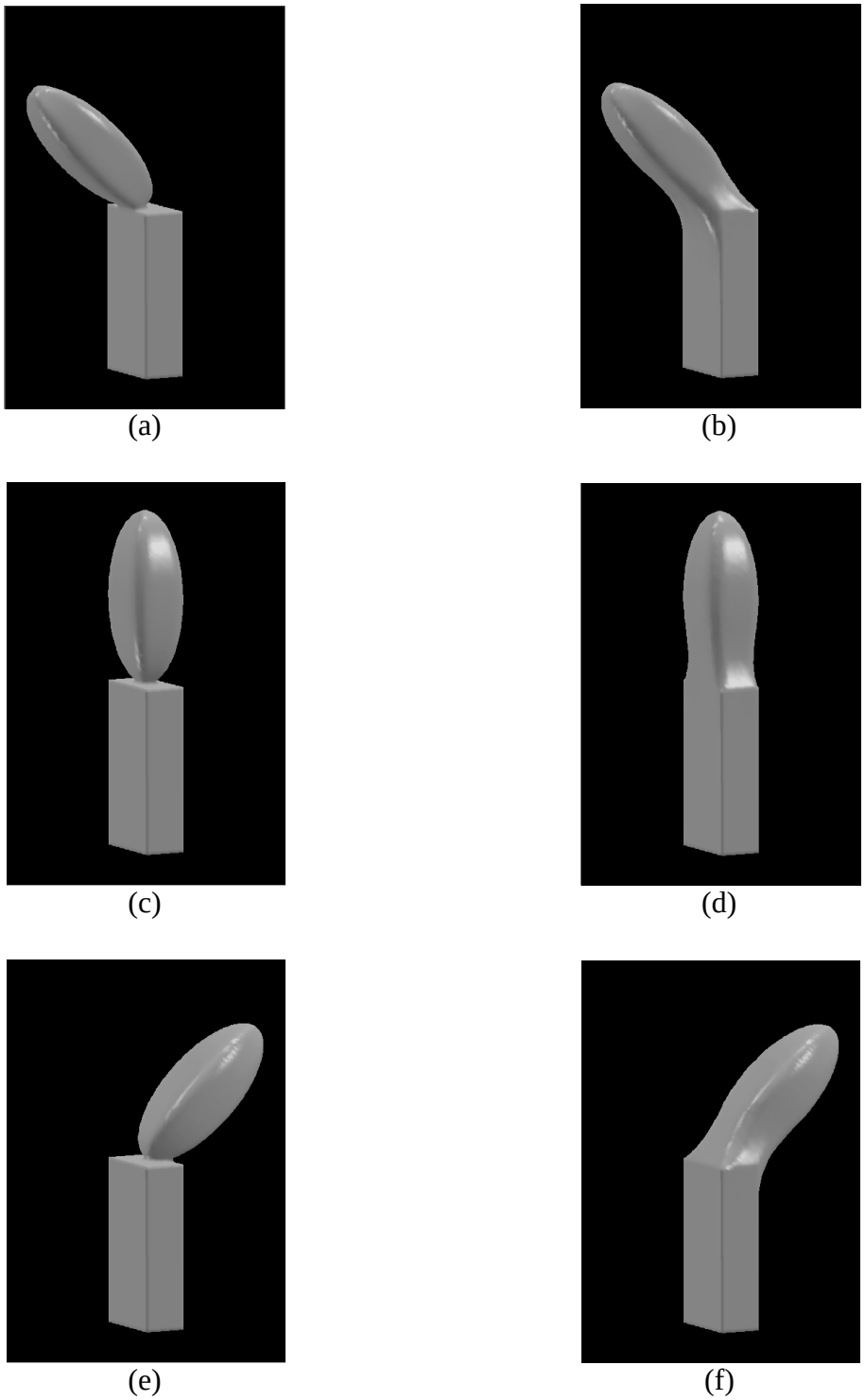


Figure 11: A short animation is created by specifying the relative motion between two superquadric components of an object. A positive-curvature flow (applied frame by frame to the joint between the two 3D models) creates a smooth, flexible object.

of sight) but from a single point of view. Range maps could come from any number of different sources including laser scanners, structured light depth systems, shape from stereo, or shape from motion. We assume that such range maps are noisy and uncertain. The goal is to combine a number of range maps from different points of view to create a 3D structure that reflects the collective confidence and depth measures.

Several examples in the literature have applied parametric models to this task. Turk and Levoy [43], for instance, "zip" together triangle meshes in order to construct 3D objects from sequences of range maps from a laser range finder. They perform minor adjustments to the surface position in order account for ambiguity in the range maps. Their approach assumes very little noise in the input, which is reasonable given the high quality of their range maps. Chen and Medioni [44] use a parametric (triangle mesh) model which expands inside a sequence of range maps. Curless and Levoy [45] describe a volume-based technique for combining range data. They use the signed distance transform to encode volume elements with data that represent the averages (with some allowance for outliers) of multiple measurements. Surfaces of objects are the level sets of volumes. Related approaches are given in [46, 47]. Bajaj et. al. [48] use a Delaunay triangulation to impose a topology on a set of unordered 3D points and then fit trivariate Bernstein-Bezier patches—i.e. a higher-order implicit model—to the data. Muraki [2] uses implicit or blobby models to reconstruct objects from range data. The individual blobs are spherically symmetric 3D potentials that are combined linearly so that they blend together. The resulting models, with approximately 400 primitives are quite coarse.

This work differs from previous work in two ways. First, rather than heuristics, our reconstruction strategy is based on a strategy that solves for the optimal surface estimate. This optimal estimate includes information about one's expectations of the likelihood of different surfaces. The result is not a closed-form solution, but an iterative process that seeks to fit a level-set model to the data while enforcing a kind of smoothness on the data.

7.3.1 Objective function for multiple range maps

The evolution equation for the estimation of optimal surfaces is shown in [42] to consist of two parts:

$$\frac{\partial \boldsymbol{x}}{\partial t} = -G(\boldsymbol{x})\boldsymbol{N} + \rho(\mathcal{S}). \tag{39}$$

This first part, $-G(\boldsymbol{x})\boldsymbol{N}$, is the data term, which is a movement with variable speed (as in expression 2 from Table 1) that is the cumulative effect from all of the individual range maps. The second part is the prior, which describes the likelihood of the surface indepen-

dent of the data. The data term is

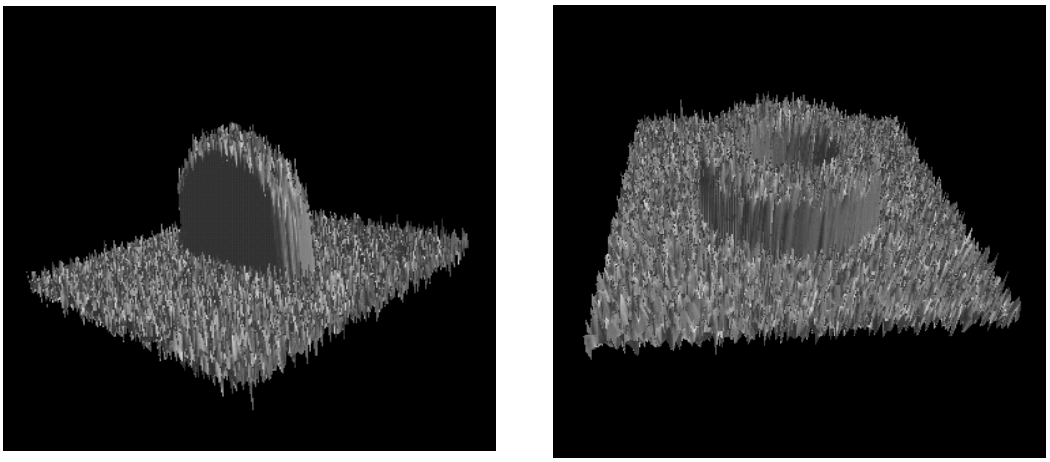
$$G(\boldsymbol{x}) = \sum_j c^{(j)}(\boldsymbol{x})\ D^{(j)}(\boldsymbol{x})\ \omega\left(D^{(i)}(\boldsymbol{x})\right)\ \gamma^{(j)}(\boldsymbol{x}), \tag{40}$$

where D_j is the signed distance along the line of sight from a range measurement in range map j associated passing through $\boldsymbol{x}$. The function $\omega : \mathbb{R} \mapsto \mathbb{R}$ is a windowing function that limits the penalty of any one range measurement, and $c(\cdot)$ is a confidence function, which is inversely proportional to the level of noise in the range measurement associated with the same line of sight. The term $\gamma(\cdot)$ is an integration constant that takes into account the curvilinear coordinate system of the range scanner.

Thus, a set of range maps creates a scalar function of 3D, which describes the movement of a surface model as it seeks the optimal surface position. In the absence of a prior, $\rho = 0$, the zero set of this function is the final position (steady state) of that evolving surface. Thus, in the absence of a prior, one could sample $g(\boldsymbol{x})$ and obtain an approximation to the optimal surface estimate. This strategy results in an algorithm that is very much like that of [45].

There are several reasons for going to an iterative scheme for finding optimal solutions. First is the use of a prior. In surface reconstruction, even a very low level of noise can degrade the quality of the rendered surfaces in the final result, and in such cases better reconstructions can be obtained by introducing a prior. Second is aliasing. Discretizing $g(\boldsymbol{x})$ and finding the zero crossings will cause aliasing in those places where the transition from positive to negative is particularly steep. A deformable model can place the surface much more precisely. The third reason for going to an iterative scheme is that despite the windowing function $\omega(\boldsymbol{x})$ there is interference between different range maps at places of high curvature. This problem is addressed by introducing a nonlinearity which is solved in an iterative scheme given by equation 39. In the work described in [21], the solution of the linear problem, the zero set of $g(\boldsymbol{x})$, serves as the initial estimate for the nonlinear, iterative optimization strategy that results from the inclusion of a prior and a nonlinear term that compensates for lack of any explicit model of self occlusions.

Equation 39 includes a *prior*, which is a likelihood function on surface shape. A reasonable choice of prior is one that models objects with less surface area as more likely than objects with more surface area. Alternatively, one could say that given a set of surfaces that are near the data, the algorithm should choose a surface that has less area. Often, but not always, this will be the smoother surface. The $\rho(\mathcal{S})$ that results from this prior is the mean curvature. Therefore the evolution of the surface, using the level-set formulation, that seeks to maximize the posterior probability (given a set of range maps and a prior that penalizes



(a) (b)

Figure 12: Range maps: Synthetic range data 200×200 pixels with 20% Gaussian white noise of a torus end (a) and side (b).

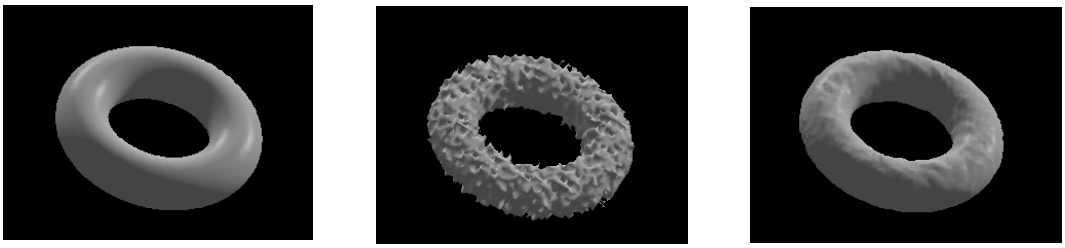
surface area) is

$$\frac{\partial \Phi(\boldsymbol{x},t)}{\partial t} = |\nabla \Phi(\boldsymbol{x})| \sum_j \left(D^{(j)}(\boldsymbol{x})\, \omega\left(D^{(i)}(\boldsymbol{x})\right) \times\ \gamma^{(j)}(\boldsymbol{x})\ c^{(j)}(\boldsymbol{x}) \frac{\left(\nabla\Phi \cdot \boldsymbol{n}^{(j)}(\boldsymbol{x})\right)^{+}}{\nabla\Phi \cdot \boldsymbol{n}^{(j)}(\boldsymbol{x})} \right) + \beta H, \tag{41}$$

where $\boldsymbol{n}^{(j)}(\boldsymbol{x})$ is the line of sight from a range finder to a 3D point, $\boldsymbol{x}$, β is a free parameter that controls the level of smoothing in the model, and H is the expression for the mean curvature given in equation 8.

Figure 12 shows a pair of simulated range maps constructed from an analytical description of a torus. These 200×200 pixel range maps are corrupted with additive Gaussian noise that has a standard deviation of 20% (as a function of the smaller of the two radii). Six synthetic noise-corrupted viewpoints of a torus are combined to create a level-set reconstruction of a torus. Figure 13(a) shows the initial model (80×80×40 voxels) used for fitting a level-set models to the range data. Figure 13(b) shows the result of the level-set models that uses 13(a) as an initial state and has a value of β equal to 0.5. The result is a reasonable reconstruction of the noiseless model (Figure 13(c)) which combines the six points of view and the smoothing function.

Figure 14(a) shows a range map taken with the Perceptron model P5000, an infra-red, time-of-flight laser range finder with a pan-tilt mechanism. Figure 14(b) shows the amplitudes associated with the return signal (an *intensity*), and 14(c) shows a surface plot of the range



(a) (b) (c)

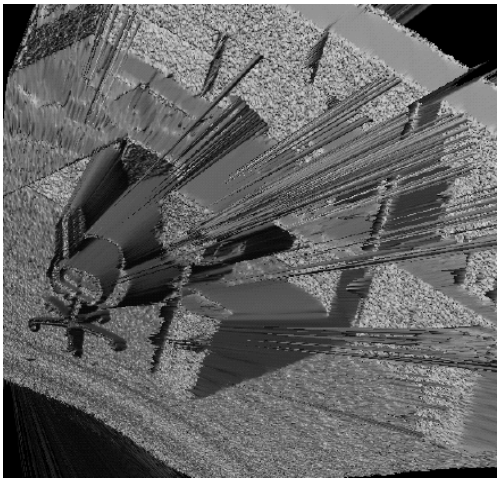
Figure 13: (a) An analytically-defined model of a torus. (b) An initial model ($80\times80\times40$ voxels) is constructed by combining six points of view of a torus and solving for $g(\boldsymbol{x}) = 0$. (c) The model, which is attracted to the range data but subject to internal forces, evolves and settles into a smoother steady state.

map to demonstrate the degree of noise (additive and outliers). Figure 14(d) shows the confidence values associated with those range measurements. These confidence values are derived from empirical data about the level of noise in the range finder (which depends on the return amplitude), and some analysis, from first principles, about the effects of uncertainty in the 3D positions of the scans and the model — which results in the lower confidence at edges as described in [42]. We combined twelve such views from different locations in the room to generate the results that follow.

Figure 15(a) shows the initial estimate based on the zero crossings of $g(\boldsymbol{x})$, and 15(b) shows the result of 32 iterations with the prior term and the correction for the surface normal direction. The size of the volume is $300 \times 150 \times 180$ voxels, and the resolution is 1.8 cm/voxel. These results show the ability of the statistically-based approach to overcome the noise in the scanner, and they show that the inclusion of iterative, model-fitting scheme helps create more accurate reconstructions. The resolution of the model falls below that of the scans, because it was limited by the random-access-memory available on our workstation. Some small features, such as the arm rests of the chairs, are lost because of the inaccuracies in the registration of the individual range maps.

(a)

(b)



(c)

(d)

Figure 14: (a) One of twelve range maps (b) The associated amplitude map (c) A surface plot of the range data to show the level of noise. (d) The confidence measures associated with those range values.

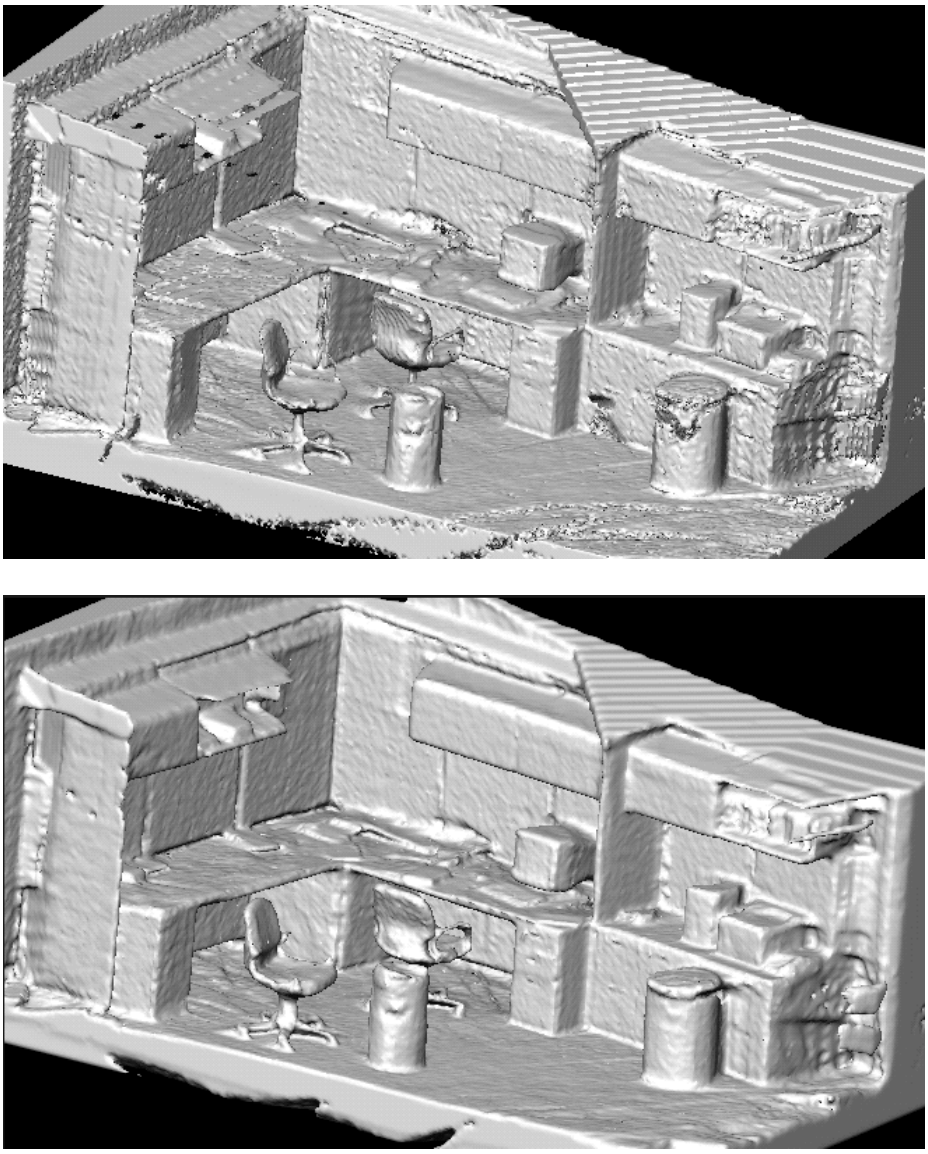


Figure 15: (top) The 3D reconstruction resulting from the zero crossings of $g(\boldsymbol{x})$ gives some averaging, but includes no prior. (bottom) The result of 32 iterations with the iterative scheme includes the prior and excludes influences of data on surfaces that face away from the scanner.

8 VISPACK

8.1 Introduction

VISPACK is a set of *C++*, object-oriented libraries for image processing, volume processing, and level-set surface modeling. It consists of five libraries: Matrix, Image, Volume, Util, and Voxmodel (level-set modeling). These libraries can be used separately or together when creating applications.

VISPACK incorporates eight basic design attributes. These are

Data Handles/Copy on Write: VISPack is an object-oriented library, and as such we allow the objects to handle memory management, and relieve the programmer (in most cases) from having to worry pointers and the corresponding memory allocation/deallocation problems. For this we use the data handles with a *copy on write protocol*. Copy constructors perform a shallow copy with reference counting until a *non const* operation on the underlying buffers forces a deep copy. Thus deep copies are performed only when necessary, but all memory is maintained by the objects and objects behave as "variables" rather than pointers.

Modified Data Hiding: Access to data in objects is generally through access methods, however, pointers to buffers for fast implementations are available.

Templates: VISPack utilizes the templating construct of C++ virtually throughout. Many of the objects, including images, volumes, lists, and arrays, are intended to support a wide range of data types. Thus, via templating programmers can define the pixels of different images of different types, such as floating point, 24-bit color, and 16-bit greyscale.

Use of Standard File Formats: When appropriate VISPack uses standard file formats. We choose formats that are well known and have publicly available libraries that can be distributed with our libraries. The matrix library uses a simple text format. The image library uses TIFF and FITS file formats. Because no standard format exists for saving volumes of data we do use a *raw* file format.

Operator Overloading: Proper use of operator overloading gives users a convenient way to execute operations on an object. When compined with the copy-on-write convention, operator overloading allows programmers to treat many heavy-weight objects (e.g. images and volumes) as variables. For instance, the following code computes non-maximal edges in a on a filtered volume.

```
Volume<float> dx, dy, dz;
Volume<float> vol_gauss = vol.gauss(0.5);
Volume<float> vol_out = (((dx = vol_gauss.dx()).power(2)
       *vol_gauss.dx(2)
     + ((dy = vol_gauss.dy()).power(2)*vol_gauss.dy(2)
     + ((dz = vol_gauss.dz()).power(2)*vol_gauss.dz(2)
     + dx*dy*(dx).dy() + dx*dz*(dx).dz())
     + dy*dz*(dy).dz()) ).zeroCrossings()
       && ((dx.power(2) + dy.power(2)) > T*T));
```

8.2 Level-Set Surface-Modeling Library

The Level-Set Surface-Modeling (LSSM) Library is an implementation of the level-set technique [10, 13] specifically for deforming surface models embedded in volumes. The implementation uses the sparse-field method described in [20]. The library implements all of the basic numerical algorithms and handles all of the data structures required to perform LSSM. The strategy for using this library is to subclass the object `VoxModel`, set some parameters, define a set of simple virtual functions that control the deformation process, initialize the model, and then direct the model to iteratively deform according to those equations. This section describes the relationship between the mathematics of previous sections and the VISPack library. Its also presents an example of using VISPack libarary to do 3D shape metamorphosis as described in Section 7.1.

8.2.1 Surface Deformation

The LSSM library allows one to solve for surface deformations, as a function of time, for general level-set surface movements of the form:

$$\frac{\partial \boldsymbol{x}}{\partial t} = \alpha \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{N}(\boldsymbol{x})) + \beta G(\boldsymbol{x}, \boldsymbol{N}(\boldsymbol{x}))\boldsymbol{N}(\boldsymbol{x}) + \gamma \boldsymbol{N}(\boldsymbol{x}) + \eta E\left(k_1(\boldsymbol{x}), k_2(\boldsymbol{x})\right), \qquad (42)$$

where $\boldsymbol{x}$ is a point on the surface. This equation is solved by representing the surface as the kth level set of an implicit function $\phi(\boldsymbol{x}, t) : \mathbb{R}^3 \times \mathbb{R}^+ \mapsto \mathbb{R}$. This gives

$$\frac{\partial \phi}{\partial t} = \alpha \boldsymbol{F}(\boldsymbol{x}, \nabla\phi)) \cdot \nabla\phi + \beta G(\boldsymbol{x}, \nabla\phi)|\nabla\phi| + \gamma|\nabla\phi| + \eta E(D\phi, \mathrm{D}^2\phi), \qquad (43)$$

where $\mathrm{D}\phi$ and $\mathrm{D}^2\phi$ are collections first and second derivatives of ϕ, respectively. This equation is solved on a discrete grid using an *up-wind* scheme gradient calculations, central differences for the curvature, and forward finite differences in time. The LSSM library uses the *sparse-field* method described in Section 6.3 and in [21].

Thus, the LSSM library offers the following capabilities:

1. Creates an initial model (with associated active set) from a volume.
2. Calculates $\Delta u^n_{i,j,k}$ and Δt using virtual functions (defined by subclasses) that describe $\boldsymbol{F}$ and G, and parameters (values set by the subclass) α, β, γ, and η.
3. Performs an update on the values of $u^n_{i,j,k}$.
4. Maintains the list of active grid points and updates the *layers* around those points in order to maintain a neighborhood from which to calculate subsequent updates.
5. Provides access to the volume that defines $u^n_{i,j,k}$ and the linked list of active grid points.

Given the volume defining $u^n_{i,j,k}$, one can then rely on the functionality of the volume library for subsequent processing, file I/O, or surface extraction.

8.2.2 Structure and Philosophy of the LSSM Library

The library is organized (mostly for ease of development) into a base class, `LevelSetModel`, and a derived class, `VoxModel`. The base class does all of the book keeping associated with the active set and surrounding *layers*, the link lists associated with those sets, and initializing the model. Thus it adds and removes voxels from the active set (and surrounding layers) in response to an update operation. The base class assumes that the subclasses know how to update individual voxels. Applications are built by subclassing `VoxModel` and redefining a small set of virtual functions that control the movement of the model.

The subclass, `VoxModel`, performs update on the grid points in the active set of the form given in Equation 18, using functions $\boldsymbol{F}$ and G and parameters α, β, γ, and η. It also calculates the maximum Δt that ensures stability. Thus a user who wishes to perform a surface deformation using the LSSM library, would create subclass of `VoxModel` and define the appropriate virtual functions and set the parameters to achieve the desired behavior.

8.2.3 The `LevelSetModel` Object

The `LevelSetModel` contains a volume of values, a volume of status flags, five lists (one active list, two inside lists, and two outside lists), and three parameters that determine the origin of the coordinate system form which the model performs its calculations.

There are two constructors, `LevelSetModel()` and `LevelSetModel( const VISVolume<float> &)`. The first simply initializes the data structure, and the second also set the values of the model volume (`_values`) to the input. Once the values have been set, one can create an initial volume from those values by calling `constructLists()`, which can also take a floating-point argument that controls the scaling of the input relative to a local distance transform near the zero set.

The list that keeps track of the active set, called `_active_list`, keeps track of the location of those grid points and a single floating-point value, which stores the change in their values from one iteration to the next.

Another important methods for users of this object is `update(float)`, which changes the grey-scale values of the grid for the active set according to the values stored in `_active_list`, and updates the status of elements on the active list as well as the values and status of nearby layers (2 inside and 2 outside). The floating point argument is the value of Δt from Equation 18, and the return value is the maximum change that occurred on the active set. Finally, the method `iterate()` calls the virtual method `calculate_change`, a virtual function which sets the values of $\Delta u^n_{i,j,k}$ and returns the maximum value of Δt for stability, and then calls `update`. For this object the function `calculate_change` performs some trivial (i.e., useless) operation.

8.2.4 The `VoxModel` Object

The `VoxModel` object is a subclass of `LevelSetModel`, and it add three things to the base class.

1. `calculate_change()` is redefined to implement the surface deformation described in Equation 43.
2. The virtual functions are declared for F (called `force`) and G (called `grow`). These functions are defined to return zero for this object.

3. The parameters that control the relative influence of the various terms are read from file by a routine `load_params`.

4. A method `rescale(float)` is defined, which resamples the volume of grid-point values into a new volume with different resolution and redefines the lists (and thereby the model) in this new volume. This method is for performing coarse-to-fine deformation procedures.

8.3 Example: 3D Shape Metamorphasis

The `Morph` object allows one to construct a sequence of volumes or surface meshes using the 3D shape metamorphasis technique described in Section 7.1, which was first proposed by Whitaker and Breen [20]. This technique relies distance transforms for both the source and target objects and uses a LSSMs to manipulate the shape of the source so that it coincides with the target. The surface deformation that describes this behavior is

$$\frac{\partial \boldsymbol{x}}{\partial t} = \beta G\left(T(\boldsymbol{x})\right) \boldsymbol{N}(\boldsymbol{x}), \tag{44}$$

where $G(\boldsymbol{x})$ is simply the distance transform (or some monotonic function thereof) of the target, and T is a coordinate transformation that aligns the source and target objects. The level-set formulation of this is

$$\frac{\partial \phi(\boldsymbol{x}, t)}{\partial t} = \beta G\left(T(\boldsymbol{x})\right) |\nabla \phi|. \tag{45}$$

The morphing process consists of several steps:

1. Read in distance transforms (in the form of volumes) for both source and target.

2. Initialize the LSSM by fitting it to the zero set of the source distance transform.

3. Update the LSSM according to Equation 45.

4. Save intermediate volumes/surfaces at regular intervals.

The remainder of this section lists the code and comments for three files, morph.h (which declares the `Morph` object), morph.C (which defines the methods) and main.C (which performs all of the I/O and uses the `Morph` object to construct a sequence of shapes.

8.4 Morph.h

```
//
// morph.h
//
//

#ifndef iris_morph_h
#define iris_morph_h

#include "voxmodel/voxmodel.h"
#include "matrix/matrix.h"

#define INIT_STATE 0
#define MORPH_STATE 1
//
// This is the morph object.  It uses all of the machinery of the base
// class to manipulate level sets.  It needs to have an initial volume
// and a final volume (which would typically be the distance transform,
// it might need a 3D transformation, and it needs to redefine the
// virtual function "grow", which takes 6 floats as input, the position
// followed by the normal vectors (all will calculated and passed into
// this method by the base class).  It might also have a state, that
// indicates whether or not it's been initialized.
//
// Functions not defined here should be defined in "morph.C"
//
class Morph: public VoxModel
{
  protected:
    VISVolume<float> _dist_source;
    VISVolume<float> _dist_target;
    VISMatrix _transform;
//
// This is the function that is used by the base class to manipulate the
level
// set.  You can define it to by anything you want.  For this object, it
will
// return a value from the distance transform of the target.
//
```

```
    virtual float grow(float x, float y, float z,
         float nx, float ny, float nz);

// There are two states.  In the first state, the model is trying to fit
// to the input data.  In this way the models starts by looking just like

// the input data
    int _state;

  public:

    Morph(const Morph& other)
    {
 _dist_target = other._dist_target;
 _initial = other._initial;
 _state = MORPH_STATE;
 _transform = VISVISMatrix(3, 3);
 _transform.identity();
// initialize();
    }

    Morph(VISVolume<float> init, VISVolume<float> d)
 :VoxModel()
    {
 _dist_target = d;
 _initial = init;
 _state = MORPH_STATE;
 _transform = VISVISMatrix(3, 3);
 _transform.identity();
// initialize();
    }

    void initialize();

// for this object I assume that the transform is just a matrix.
// but it could be anything
    void transform(const VISVISMatrix& t)
    { _transform = t;}

    const VISVISMatrix& transform()
    { return(_transform);}
```

```
    void distance(const VISVolume<float> d)
    { _dist_target = d;}
    VISVolume<float> distance()
    { return(_dist_target);}

};
#endif
```

8.5 Morph.C

```
#include "morph.h"
#include "util/geometry.h"
#include "util/mathutil.h"


//
// this is the virtual function, that  is the guts of it all.
//

float Morph::grow(float x, float y, float z,
    float nx, float ny, float nz)
{

// this says you are in the morph state (things have been initialized)
    if (_state == MORPH_STATE)
 {
 float xx, yy, zz;
 VISPoint p(4u);
 p.at(0) = x;
 p.at(1) = y;
 p.at(2) = z;
 p.at(3) = 1;
 VISPoint p_tmp;
// this is where you could put some other transform.
 p_tmp = _transform*p;

 xx = p_tmp.x();
 yy = p_tmp.y();
```

```
 zz = p_tmp.z();

// make sure you are not out of the bounds
// of your distance volume.
 if (_dist_target.checkBounds(xx, yy, zz))
// if not, get the distance (use trilinear interpolation).
     return(_dist_target.interp(xx, yy, zz));
 else
     return(0.0f);
    }
else
    {
// if you are still initializing, then move toward the zero set of
// your initial case
 if (_initial.checkBounds(x, y, z))
     return(_initial.interp(x, y, z));
 else
     return(0.0f);
    }
}

// this makes the model look like the input.
#define INIT_ITERATIONS 5
void Morph::initialize()
{
    _values = _initial;
    int state_tmp = _state;
    _state = INIT_STATE;
    construct_lists(DIFFERENCE_FACTOR);
// these couple of iterations are required to make sure that the zero
// sets of the model match the zero sets of the
//
    for (int i = 0; i < INIT_ITERATIONS; i++)
 {
// limit the dt to 1.0 so that the model settles in to a solution
     update(::min(calculate_change(), 1.0f));
 }
    _state = state_tmp;
}
```

8.6 Main.C

```
#include "vol/volume.h"
#include "vol/volumefile.h"
#include "image/imagefile.h"
#include "morph.h"
#include <string.h>


const int V_HEIGHT = (40);
const int V_WIDTH = (40);
const int V_DEPTH = (40);

#define XY_RADIUS (12)  // this matches the 2.5D data generated in
torus.C
#define T_RADIUS (4)  // this matches the 2.5D data generated in torus.C
#define S_RADIUS (12)  // radius of a sphere

#define B_WIDTH (20.0f)
#define B_HEIGHT (60.0f)
#define B_DEPTH (20.0f)

#define B_CENTER_X (12.0f)
#define B_CENTER_Y (32.0f)
#define B_CENTER_Z (12.0f)

float sphere(unsigned x, unsigned y, unsigned z);
float torus(unsigned x, unsigned y, unsigned z);
float cube(unsigned x, unsigned y, unsigned z);


// This is a program that does the morph.  If you give it two
// arguments, it reads the initial model and the dist trans for the
// final model from the two file names given, otherwise, it makes a
sphere
// and deforms it into a torus

main(int argc, char** argv)
{
```

```
VISVolume<float> vol_source, vol_target;
VISVolumeFile vol_file;
int i;
char fname[80];

vol_source = VISVolume<float>(25,65,25);
vol_source.evaluate(cube);

if (argc > 2)
    {
// read in the sourceing model
 vol_source = VISVolume<float>(vol_file.read_float(argv[1]));
// read in the dist trans of the final model
     vol_target = VISVolume<float>(vol_file.read_float(argv[2]));
    }
else
// make up some volumes
    {
 vol_source = VISVolume<float>(V_WIDTH, V_HEIGHT, V_DEPTH);
 vol_source.evaluate(sphere);
 vol_target = VISVolume<float>(V_WIDTH, V_HEIGHT, V_DEPTH);
 vol_target.evaluate(torus);
    }

// create morph object
Morph morph(vol_source, vol_target);
// loads in some parameters (for morphing these are all zero but one)
// i.e.
//
//
//
//
morph.load_parameters("morph_params");
morph.initialize();
vol_file.write_float(morph.values(), "morph0.flt");

float dt;

// do 150 iterations for your model to get from start to finish
// probably don't need this many iterations
```

```
for (i = 0; i < 150; i++)
    {
 dt = morph.calculate_change();
// limit dt to 0.5 so that model never overshoots goal
 dt = min(dt, 0.5f);
 morph.update(dt);

 printf("iteration %d dt %f\n", i, dt);

 if (((i + 1)%10) == 0)
 {
// save every tenth volume
     sprintf(fname, "morph_out.%d.dat", i + 1);
     vol_file.write_float(morph.values(), fname);
 }
    }

// save a surface model (i.e. marching cubes).
vol_file.march(0.0f, morph.values(), ‘‘morph_final.iv’’);

printf("done\n");

}
```

References

[1] J. Blinn, "A generalization of algebraic surface drawing," *ACM Trans. on Graphics*, vol. 1, pp. 235–256, March 1982.

[2] S. Muraki, "Volumetric shape description of range data using "blobby model"," in *SIGGRAPH '91 Proceedings* (T. W. Sederberg, ed.), pp. 227–235, July 1991.

[3] G. Taubin, "An accurate algorithm for rasterizing algebraic curves and surfaces," *IEEE Computer Graphics & Applications*, March 1994.

[4] D. Breen, S. Mauch, and R. Whitaker, "3d scan conversion of csg models into distance, closest-point and colour volumes," in *Volume Graphics* (M. Chen, A. Kaufman, and R. Yagel, eds.), pp. 135–158, London: Springer, 2000.

[5] W. Lorenson and H. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1982.

[6] M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, vol. 9, no. 3, pp. 245–261, 1990.

[7] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *SIGGRAPH '88 Proceedings*, pp. 65–74, August 1988.

[8] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, pp. 321–323, 1987.

[9] D. Terzopoulos and K. Fleischer, "Deformable models," *The Visual Computer*, vol. 4, pp. 306–331, December 1988.

[10] S. Osher and J. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Jrnl. of Comp. Phys.*, vol. 79, pp. 12–49, 1988.

[11] J. Sethian, *Level Set Methods and Fast Marching Methods*. Cambridge: Cambridge University Press, second ed., 1999.

[12] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci.*, vol. 93, no. 4, pp. 1591–1595, 1996.

[13] J. A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Material Sciences*. Cambridge University Press, 1996.

[14] S. Osher and R. Fedkiw, "Level set methods: An overview and some recent results," Tech. Rep. 00-08, UCLA Center for Applied Mathematics, Department of Mathematics, University of California, Los Angeles, 2000.

[15] L. Alvarez and J.-M. Morel, "A morphological approach to multiscale analysis: From principles to equations," in *Geometry-Driven Diffusion in Computer Vision* (B. M. ter Haar Romeny, ed.), pp. 4–21, Kluwer Academic Publishers, 1994.

[16] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," in *Fifth Int. Conf. on Comp. Vision*, pp. 694–699, IEEE, IEEE Computer Society Press, 1995.

[17] B. B. Kimia and S. W. Zucker, "Exploring the shape manifold: the role of conservation laws.," in *Shape in Picture: the mathematical description of shape in greylevel images* (Y.-L. O, A. Toet, H. Heijmans, D. H. Foster, and P. Meer, eds.), Springer-Verlag, 1992.

[18] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 2, pp. 158–175, 1995.

[19] R. T. Whitaker and D. T. Chen, "Embedded active surfaces for volume visualization," in *SPIE Medical Imaging 1994*, (Newport Beach, California), 1994.

[20] R. Whitaker and D. Breen, "Level-set models for the deformation of solid objects," in *The Third International Workshop on Implicit Surfaces*, pp. 19–35, Eurographics, 1998.

[21] R. T. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int. Jrnl. of Comp. Vision*, vol. October, no. 3, pp. 203–231, 1998.

[22] R. T. Whitaker, "Algorithms for implicit deformable models," in *Fifth Intern. Conf. on Comp. Vision*, IEEE, IEEE Computer Society Press, 1995.

[23] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi, "Gradient flows and geometric active contour models," in *Fifth Int. Conf. on Comp. Vision*, pp. 810–815, IEEE, IEEE Computer Society Press, 1995.

[24] A. Yezzi, S. Kichenassamy, A. Kumar, P. Olver, and A. Tannenbaum, "A geometric snake model for segmentation of medical imagery," *IEEE Transactions on Medical Imaging*, vol. 16, pp. 199–209, April 1997.

[25] L. Lorigo, O. Faugeraus, W. Grimson, R. Keriven, and R. Kikinis, "Segmentation of bone in clinical knee MRI using texture-based geodesic active contours," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI '98)* (W. Wells, A. Colchester, and S. Delp, eds.), pp. 1195–1204, October 1998.

[26] S. Osher and J. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Jrnl. of Comp. Phys.*, vol. 79, pp. 12–49, 1988.

[27] X. Xue and R. Whitaker, "Variable-conductance, level-set curvature for image denoising," in *IEEE International Conference on Image Processing*, p. To Appear., October 2001.

[28] D. Adalstein and J. A. Sethian, "A fast level set method for propagating interfaces," *Jrnl. of Comp. Phys.*, pp. 269–277, 1995.

[29] D. Breen and R. Whitaker, "A level-set approach to 3D shape metamorphosis," *IEEE Transactions on Visualization and Computer Graphics*, p. To Appear., 2001.

[30] A. Lerios, C. D. Garfinkle, and M. Levoy, "Feature-Based volume metamorphosis," in *SIGGRAPH '95 Conference Proceedings* (R. Cook, ed.), Annual Conference Series, pp. 449–456, Addison Wesley, Aug. 1995.

[31] J. Kent, W. Carlson, and R. Parent, "Shape transformation for polyhedral objects," in *SIGGRAPH '92 Proceedings*, pp. 47–54, July 1992.

[32] J. Rossignac and A. Kaul, "AGRELS and BIPs: Metamorphosis as a bezier curve in the space of polyhedra," *Computer Graphics Forum (Eurographics '94 Proceedings)*, vol. 13, pp. C–179–C–184, September 1994.

[33] J. F. Hughes, "Scheduled Fourier volume morphing," in *Computer Graphics (SIGGRAPH '92 Proceedings)* (E. Catmull, ed.), vol. 26, pp. 43–46, July 1992.

[34] B. Payne and A. Toga, "Distance field manipulation of surface models," *IEEE Computer Graphics and Applications*, vol. 12, no. 1, pp. 65–71, 1992.

[35] D. Cohen-Or, D. Levin, and A. Solomivici, "Three-dimensional distance field metamorphosis," *ACM Transactions on Graphics*, vol. 17, no. 2, pp. 116–141, 1998.

[36] P. Getto and D. Breen, "An object-oriented architecture for a computer animation system," *The Visual Computer*, vol. 6, pp. 79–92, March 1990.

[37] D. Breen, S. Mauch, and R. Whitaker, "3D scan conversion of CSG models into distance volumes," in *Proceedings of the 1998 Symposium on Volume Visualization*, pp. 7–14, ACM SIGGRAPH, October 1998.

[38] A. Middleditch and K. Sears, "Blend surfaces for set theoretic volume modeling systems," in *SIGGRAPH '85 Proceedings*, pp. 161–170, July 1985.

[39] J. Bloomenthal and K. Shoemake, "Convolution surfaces," in *SIGGRAPH '91 Proceedings* (T. W. Sederberg, ed.), pp. 251–257, July 1991.

[40] M. Desbrun and M.-P. Gascuel, "Animating soft substances with implicit surfaces," in *SIGGRAPH '95 Proceedings*, pp. 287–290, August 1995.

[41] R. T. Whitaker, "Volumetric deformable models: Active blobs," in *Visualization In Biomedical Computing 1994* (R. A. Robb, ed.), (Mayo Clinic, Rochester, Minnesota), pp. 122–134, SPIE, 1994.

[42] R. T. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int. Jrnl. of Comp. Vision*, vol. October, no. 3, pp. 203–231, 1998.

[43] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proc. of SIGGRAPH '94*, pp. 311–318, ACM SIGGRAPH, August 1994.

[44] Y. Chen and G. Médioni, "Fitting a surface to 3-D points using an inflating ballon model," in *Second CAD-Based Vision Workshop* (A. Kak and K. Ikeuchi, eds.), vol. 13, pp. 266–273, IEEE, 1994.

[45] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. of SIGGRAPH '96*, pp. 303–312, ACM SIGGRAPH, August 1996.

[46] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *Computer Graphics*, vol. 26, no. 2, pp. 71–78, 1992.

[47] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt, "Reliable surface reconstruction from multiple range images," in *Euro. Conf. on Comp. Vision*, Springer-Verlag, 1996.

[48] C. Bajaj, F. Bernardini, and G. Xu, "Automatic reconstruction of surfaces and scalar fields from 3D scans," in *SIGGRAPH '95 Proceedings*, pp. 109–118, August 1995.

Module 4

Level Set Applications II

A Level-Set Approach to 3D Reconstruction From Range Data

Ross T. Whitaker
School of Computing
University of Utah
Salt Lake City, UT 84112-9205
email: whitaker@cs.utah.edu

Abstract

This paper presents a method that uses the level sets of volumes to reconstruct the shapes of 3D objects from range data. The strategy is to formulate 3D reconstruction as a statistical problem: find that surface which is mostly likely, given the data and some prior knowledge about the application domain. The resulting optimization problem is solved by an incremental process of deformation. We represent a deformable surface as the level set of a discretely sampled scalar function of 3 dimensions, i.e. a volume. Such *level-set models* have been shown to mimic conventional deformable surface models by encoding surface movements as changes in the greyscale values of the volume. The result is a voxel-based modeling technology that offers several advantages over conventional parametric models, including flexible topology, no need for reparameterization, concise descriptions of differential structure, and a natural scale space for hierarchical representations. This paper builds on previous work in both 3D reconstruction and level-set modeling. It presents a fundamental result in surface estimation from range data: an analytical characterization of the surface that maximizes the posterior probability. It also presents a novel computational technique for level-set modeling, called the sparse-field algorithm, which combines the advantages of a level-set approach with the computational efficiency and accuracy of a parametric representation. The sparse-field algorithm is more efficient than other approaches, and because it assigns the level set to a specific set of grid points, it positions the level-set model more accurately than the grid itself. These properties, computational efficiency and sub-cell accuracy, are essential when trying to reconstruct the shapes of 3D objects. Results are shown for the reconstruction objects from sets of noisy and overlapping range maps.

Appears in: *The International Journal of Computer Vision* , 29(3), October, 1998, pp. 203–231.

1 Introduction

There are two aspects to this research. The first aspect is the application, which is the development of 3D models from range data. That problem is formulated using a statistical approach, which maximizes the posterior probability of a surface. The result is a surface that is described by a nonlinear objective function. That optimization is solved using a variational approach and a gradient-descent method. This optimization produces an evolution equation for deforming a surface so that it matches a given set of data. The work in this paper uses the level sets of volumes as a means of representing and manipulating object shapes. The second aspect of the research is the enhancement of the underlying level-set technology in order to make it suitable for problems in 3D reconstruction. This section introduces the issues surrounding those two aspects of the work, and lays out the overall structure of the paper.

1.1 3D Reconstruction

When investigating 3D reconstruction, it is necessary to describe the kind of data being considered. This work deals primarily with data that is from range or distance sensors that have some kind of "scanning capability". Three different properties of this data are important. First, the data is range or distance (i.e., direct 3D) compared to intensity or luminance, as one would expect from an X-ray or a video camera. Second, the data is relatively dense, rather than a sparse scattering of 3D points. This density is obtained by rotating or translating a device that takes a discrete set of 3D measurements. From a single point of view the range finder sweeps out a volume, and there is a 2D topology on these range measurements that is induced by the motion of the scanning device. The third important aspect of this data is that it is noisy. Thus, the 3D reconstruction problem is not strictly geometric; it is also statistical. For this work we assume that algorithms for calibrating the range sensor and registering the multiple views are taken from the variety of automated and semiautomated techniques that are in the literature — e.g. (Besl and McKay 1992, Chen and Medioni 1992, Zhang 1994).

Under the circumstances described above, approaches to 3D reconstruction can be distinguished based on the "level" of the models that one uses. High-level models have been used extensively in computer vision — some example are given in (Jain and Jain 1990, Jain and Flynn 1993). High-level models are those that represent specific objects or classes of objects. Such approaches can work with relatively little information (such as sparse sets of features), and when they work they can lead directly to higher level processes such as recognition. High-level approaches usually work best in situations where the domain is restricted and well characterized, and where the distinguishing shape characteristics of a particular object can be captured by relatively few parameters. Typically, such reconstructions do not capture those details that are unique to a particular object.

Another class of approaches could be characterized as "mid level". Mid-level approaches to reconstruction are characterized by the use of geometric primitives that can be combined in various ways to form more complicated objects. The assumption is that the objects in the domain can be decomposed into finite set of such primitives. This strategy can work quite well on certain domains, such as man-made objects—many of which were designed using such primitives. The fitting of primitives can also provide important structural information such as part/sub-part decomposition. The results of such systems are generally some mixture of the data and the models, but they tend to strongly reflect the shapes of the models or primitives that are chosen.

Finally, one could characterize the class of "low level" approaches to 3D reconstruction. These approaches use a few general assumptions about the domain; assumptions are often at the level of basic surface geometry, e.g. continuity. Low-level approaches can capture more detail but often provide very little higher level information. For this reason such approaches are well suited for applications that are geared to visualization by a human, rather than recognition by some automated system. These approaches form surfaces from sets of 3D data by performing a fusion of different scans, but they typically impose very little structure on the data. The performance of low-level approaches tends to degrade gradually as the assumptions about the environment and the sensor are violated. However, low-level approaches typically require more data in order to give useful results. The voxel-based approach in this paper is a low-level approach and is therefore best suited to applications that provide a relatively large amount of data that must be fused to produce a result with a great amount of detail.

1.2 Level-set models

The strategy taken within this research is to pose the reconstruction problem as the process of finding the surface or set of surfaces that are most likely to have given rise to the data. Thus, it is a Maximum A Posteriori (MAP) strategy which can combine the data with known properties or tendencies of the surfaces being measured. Finding those surfaces with the highest likelihood is generally a nonlinear optimization problem.

A common and often effective strategy for solving such optimization problems is to use a variational approach. That is, start with an initial solution and perturb or deform that solution so that it improves the overall likelihood. Thus, the variational approach requires a modeling technology that can undergo such goal-driven deformations. Such models, often called *deformable models*, are used extensively in the computer graphics and computer vision literature, e.g. (Kass, Witkin and Terzopoulos 1987, Staib and Duncan 1992, Miller, Breen, Lorensen, O'Bara and Wozny 1991, Terzopoulos, Witkin and Kass 1988). These models typically undergo evolution processes which implement some type of hill-climbing strategy on the objective function.

Conventional geometric models are parametric. That is, they are mappings from one space, which coincides with the dimensionality of the model, to another, which is the range. Typically, these parameterizations rely on a set of basis functions that span some finite subspace of all possible shapes. Thus when considering deformations, parametric models have several drawbacks which make them inadequate for certain kinds of applications. The dependency on the parameterization and an associated basis is critical; it limits the kinds of shapes a model can represent. Models typically do not deform far from their initial conditions without some sort of reparameterization. Such reparameterizations are often inefficient and developed on the basis of heuristics rather than a consistent mathematical foundation. The parameterization can also make it difficult to measure the intrinsic geometry of the model, as with polygonal meshes for instance.

An alternative to a parametric model is an implicit model, i.e., specifying a model as a level set of a scalar function, ϕ. This scalar function can be sampled on a discrete rectilinear grid. Such a *level-set* representation (Sethian 1996) has a number of practical and theoretical advantages over conventional parametric models, especially in the context of deformation and reconstruction. First, level-set models are topologically flexible, that is, the models can "split" into pieces to form multiple objects (Malladi, Sethian and Vemuri 1995, Whitaker and Chen 1994). Second the evolution of the embedding ϕ is a differential expression that is invariant to orthogonal group transformations (rotations and translations). The shapes formed by the level sets of ϕ are restricted only by the resolution of the discrete sampling used to represent ϕ. Finally, the representation of deformable models in terms of a *multidimensional image* provides a method for multi-scale or multi-grid solutions. For instance solutions can start on a relatively coarse grid and then proceed to progressively finer grids as the energy reaches a minimum (Whitaker 1995). Such a multigrid strategy reduces computation time, controls the relative importance of large and small-scale structures in the input image, and helps to simplify the objective function.

Despite these advantages, level-set models suffer from several drawbacks compared with parametric models. One disadvantage is the large number of computations needed to solve these equations over the entire range. Surface deformations in 3D, for example, require solutions to a 3D, nonlinear, partial differential equation: a significant computational task. Another drawback of level-set models is the absence of any direct, efficient representation of the surface during deformation. Such direct representations are useful when incorporating forces that depend on the multi-local or global information. A third drawback is the finite resolution that is imposed by the discrete approximation to the scalar field. Level-set models are limited by resolution rather than shape.

These problems are significant in the context of 3D reconstruction. The sparse-field algorithm described in this paper addresses these issues by representing a surface as both a discretely-sampled 3D field and a subset of that field which consists of a set of grid points (or voxels), called *active points*, through which the model passes. At each time step in the deformation process only a thin layer of voxels near the active points are visited and updated. In this way the computational complexity of the algorithm grows with the dimensionality of the surface, rather than the dimensionality of the volume. The set of active points are kept in a linked list which enables quick and efficient access to a set of points near or on the model.

The remainder of this paper proceeds as follows. After a brief review in Section 2 of other relevant work, Sec-

tion 3 presents the formulation of 3D reconstruction as a process of finding the most likely surface given a set of range data and some general knowledge about the relative likelihoods of different surface properties. Solving that problem requires the ability to manipulate or deform surface shape in a systematic manner. Section 4 shows how this deformation can be achieved via an implicit representation, using the level sets of a discretely-sampled scalar function of 3D. That section discusses the numerical methods that are used to solve the equations of motion. Section 5 describes an efficient computational technique, the sparse-field algorithm, and presents the results of an empirical analysis which shows that the sparse-field algorithm gives solutions that compare favorably to previously proposed algorithms. Additionally, the sparse-field algorithm is shown to overcome the aliasing artifacts associated with other level-set approaches and thereby achieves greater accuracy compared to those approaches. Section 6 shows how the level-set technology can be applied to the problem of 3D reconstruction. It shows how using level-set models within the MAP reconstruction framework provides new capabilities for surface reconstruction.

2 Related work

Several low- and mid-level reconstruction approaches described in the literature are worth noting. Turk and Levoy (1994) describe a "zippering" algorithm that combines triangle meshes, each representing a range map of the same object from a different point of view. Chen and Médioni (1994) use a triangle mesh which expands inside a sequence of range maps. This strategy is similar to that of Miller et al. (1991) which was demonstrated on 3D medical data.

Several low-level volumetric approaches have been proposed. Chien and Aggarwal (1989) have used binary volumes with the aid of oct-trees to reconstruct objects from multiple silhouettes. The focus is on using silhouettes and object features to do recognition. Muraki (1991) uses implicit or blobby models to reconstruct objects from range data. The individual blobs are spherically symmetric 3D potentials that are combined linearly so that they blend together. The global nature of the potentials makes updates somewhat expensive, thus limiting the number of primitives that can be efficiently computed.

Hoppe, DeRose, Duchamp, McDonald and Stuetzle (1992) proposed an implicit strategy which constructs a 3D field based on the signed distance transform of tangent planes generated from unordered 3D data. Regions are always associated with the nearest points, and therefore there is little or no averaging. Curless and Levoy (1996) describe a volume-based technique for combining range data. They use the signed distance transform to encode volume elements with data that represent the averages (with some allowance for outliers) of multiple measurements. Surfaces of objects are the level sets of volumes. They show that the resolution of the scanner can be overcome by such an averaging technique. Hilton, Stoddart, Illingworth and Windeatt (1996) describe a similar algorithm which is an extension to in (Hoppe et al. 1992) that accounts for interference at occluding contours and thin objects.

Another volumetric approach, which is more of a "mid-level" strategy, is that of (DeCarlo and Metaxas 1995), in which they use a combination of primitives that can deform, split, and even change topology in order to match the input data. The computer vision literature shows numerous mid-level approaches that fit volumetric models to range data—see (Dickinson, Metaxas and Pentland 1997), for example.

In robotics several researchers (Elfes 1989, Moravec 1988) have investigated the use of *occupancy grids*. The sensor model and the methods for combining sensor measurements from different points of view follow from a Bayesian formulation. Strictly speaking an occupancy grid does not give a 3D reconstruction; occupancies do not give the most likely surfaces.

With regard to deformable models, there are several parametric approaches that seek to overcome the usual topological limitations of parametric models. DeCarlo and Metaxas (1995) allow the primitives associated with an object to make discrete changes as a result of a set of rules and thresholds. Szeliski, Tonnesen and Terzopoulos (1993) propose systems of oriented particles that join together, disconnect, and rejoin in order to change topology. McInerny and Terzopoulos (1995) propose parametric models that are reparameterized based on a local grid structure that allows for changes in topology. Despite these advances, for 3D reconstruction the level-set approach offers several practical advantages, which are discussed in more detail in the sections that follow.

With regard to level-set models, this work draws on a number of recent developments in computational physics

and computer vision. Osher and Sethian (1988) have proposed the embedding of wavefront propagations that model physical systems, and they have developed numerical schemes, called *up-wind* schemes, to solve the resulting equations. In image processing Alvarez, Guichard, Lions and Morel (1992) have proposed a morphological scale space for planar curves which relies on geometric invariant curve evolutions. In computer vision Kimia, Tannenbaum and Zucker (1992) have proposed a reaction-diffusion space in which singularities represent basic properties of 2D shape. They use the same strategy as (Osher and Sethian 1988) of embedding planar curves in order to create geometric flows that are independent of any particular parameterization.

Malladi et al. (1995) have proposed a segmentation scheme based on a wavefront propagation that allows seed points to contract or expand. The author (Whitaker and Chen 1994) has shown that image "forces" (the term used in the sense of first-order physics) can drive level sets toward interesting features in images or volumes, as with conventional deformable models, while geometric flows can be used to enforce smoothness and continuity. Caselles, Kimmel and Sapiro (1995) have reached a similar formulation using a conformal mapping and have shown that the equations resulting from moving level sets in this manner are well posed. Adalstein and Sethian (1995) have proposed a narrow-band scheme, with a finite band of 6-12 grid points on either side of the level set, for reducing the computations associated with level-set models. Results in successive sections of this paper will show the ability to reduce the width of this narrow band to a single grid point will improve computational performance and accuracy.

This paper makes several contributions to previous work in this field. One is a statistical formulation of the 3D reconstruction problem that gives an optimal surface estimate while making very few assumptions about the scene. This formulation incorporates a noise model and may include some prior knowledge about the relative likelihoods of different kinds of surfaces. The result is quite general, but it is particularly well suited for level-set models. This paper presents examples that show effectiveness of the level-set approach for this application. Another contribution is the numerical scheme for implementing the level-set models. This new scheme compares favorably with previously proposed methods both in terms of computational efficiency and accuracy.

3 A MAP Reconstruction Strategy

A range finder is a device that produces a distance measurement along a particular line of site. The distance measurement and, to a lesser degree, the direction of the line of sight are corrupted by noise. For the purposes of this work we assume that the noise is additive and Gaussian. When the range finder is combined with a scanning mechanism, it produces a 2D array of data, and the noise is often assumed to be independent from one element of the array to another. In the event that the noise is non-stationary, we assume that the variability of the noise process is known (e.g. as a function of object distance or scanner orientation). The scanning mechanism is typically calibrated, enabling one to calculate the direction of the line of sight associated with each measurement in the 2D array. In the event that the scanning mechanism does not produce a 2D array, but rather an unordered list of range measurements taken in different directions, one can impose a 2D topology on the data by the use of nearest-neighbor relationships such as Delaunay triangulation. A single 2D array of range data taken from a single location of the scanner is called a *range map*. One of the goals of this work is to combine the information from a collection of range maps.

The above sensor model has some significant shortcomings. First, the noise is rarely additive Gaussian. Outliers, for instance, are a problem. The noise can often be related to surface characteristics, and therefore could be correlated. In the case of structured light sensors, such as the one used for some of the results in this paper, the data contains occluded regions for which there is no data at all. As a theoretical consideration, Gaussian noise allows finite probabilities for measurements behind the scanner. In cases where the noise level is some appreciable fraction of the the distance from the scanner (very rare in practice), a log-normal distribution for the noise might be more appropriate.

In this work, we start with the simple sensor model, i.e. independent, additive, Gaussian noise, to generate the basic algorithm and then modify that algorithm to account for some of these other complicating factors. The outliers, for instance, are handled within the minimization process rather than the underlying model. Areas where the sensor failed (either by lack of signal or occlusion) are handled by giving such points low confidence, i.e. large variance, so that they have little effect on the final results. The formulation that results from these assumptions is optimal for

independent, additive, Gaussian noise, but later sections will show that it is useful when the noise is more complex (such as in the case of outliers).

The strategy in this work is to use a maximum a posteriori (MAP) approach to construct an expression for the surface likelihood conditional on the measured data. The total error associated with a model converts into a volume integral, and the Euler-Lagrange of that volume integral defines the motion of a deformable model that seeks to maximize this likelihood.

Let $\mathcal{S}$, a compact subset of 3D, be the surface that encloses the object Ω, i.e., $\mathcal{S} = \partial\Omega$. Let $\{R^{(1)}, \ldots, R^{(n)}\}$ be a set of range maps, each of which consists of set of 3D points $R^{(i)} = \{\mathbf{r}_1^{(i)}, \ldots, \mathbf{r}_m^{(i)}\}$, which can arranged in a 2D grid, with coordinates u, v, defined by the scanning mechanism. Associated with each grid point is a ray, called the line of sight, along which the range measurement is taken. Assume that the rays within a single range map do not intersect, as is the case with the spherical, cylindrical, or linear projective scanning mechanisms that are used on most range finders.

The posterior probability of $\mathcal{S}$ is given by Bayes rule:

$$P(\mathcal{S}|R^{(1)}, \ldots, R^{(n)}) = \frac{P(R^{(1)}, \ldots, R^{(n)}|\mathcal{S})P(\mathcal{S})}{P(R^{(1)}, \ldots, R^{(n)})}. \tag{1}$$

The goal is to find the most likely surface model $\mathcal{S}$ to give rise to a particular collection of range data,

$$\sup_{\mathcal{S}} \left[P(\mathcal{S}|R^{(1)}, \ldots, R^{(n)})\right] = \sup_{\mathcal{S}} \left[P(R^{(1)}, \ldots, R^{(n)}|\mathcal{S})P(\mathcal{S})\right] \tag{2}$$

where the denominator $P(R^{(1)}, \ldots, R^{(n)})$ is removed because it is a normalization factor that does not depend on $\mathcal{S}$. The term $P(\mathcal{S})$ is the prior, which reflects the fact that within the set of possible surfaces, some are more likely than others, independent of the data.

The range maps, conditional on the surface position, are independent random variables (this ignores the effects of surface properties on error, as discussed above), and thus,

$$P(R^{(1)}, \ldots, R^{(n)}|\mathcal{S}) = P(R^{(1)}|\mathcal{S}) \ldots P(R^{(n)}|\mathcal{S}) = \prod_i P(R^{(i)}|\mathcal{S}). \tag{3}$$

Each $R^{(i)}$ consists of an array of 3D range data that can be represented as $\mathbf{r}_{u,v}^{(i)}$. Within a given range map each range measurement is a independent random variable, and therefore

$$P(R^{(1)}, \ldots, R^{(n)}|\mathcal{S}) = \prod_i \prod_{u,v} P(\mathbf{r}_{u,v}^{(i)}|\mathcal{S}). \tag{4}$$

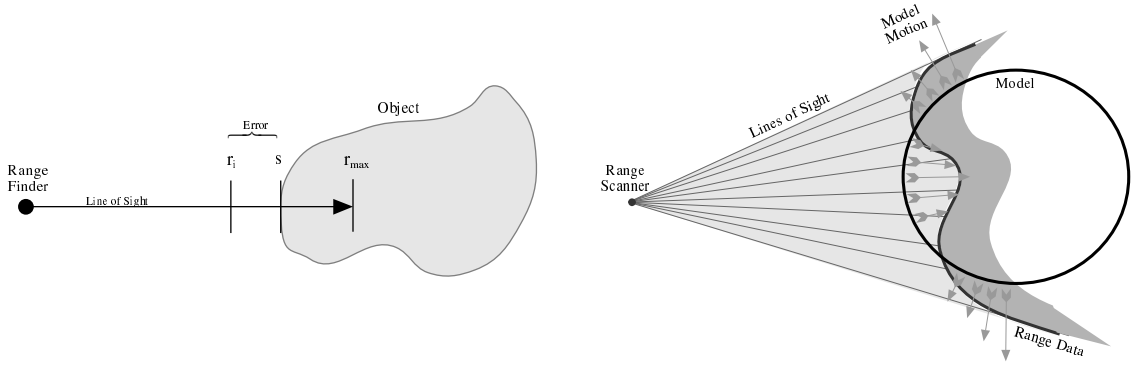
As is typical with such MAP algorithms, the maximization is performed as a minimization on the negative logarithm of the probability:

$$\sup_{\mathcal{S}} \left\{\ln\left(P(\mathcal{S}|R^{(1)}, \ldots, R^{(n)})\right)\right\} = \inf_{\mathcal{S}} \left\{-\sum_i \sum_{u,v} \ln P(\mathbf{r}_{u,v}^{(i)}|\mathcal{S}) - \ln P(\mathcal{S})\right\}. \tag{5}$$

The right-hand side of equation (5) reflects the combination of terms that is typical with MAP reconstructions. The terms are of two different types: those terms that enforce the solution to look like the data and those terms that enforce the prior, i.e., that encourage the solutions to conform to those expectations that are independent of the data.

Each individual range measurement is dependent not on the surface $\mathcal{S}$ as a whole, but on that particular point on the surface that is first intersected along the line of sight from the scanner location. Consider a single range reading r_i, represented as a distance along its associated line of sight. Suppose that the distance to the surface along that line of sight is given by s_i. The Gaussian noise model gives the following probability density function for individual range readings,

$$-\ln P(r_i|\mathcal{S}) = \frac{1}{\sigma_i^2}(s_i - r_i)^2 + \frac{1}{\sqrt{2\Pi\sigma}} = c_i(s_i - r_i)^2 + k(c_i), \tag{6}$$



(a) (b)

Figure 1: The MAP reconstruction strategy: (a) The squared error along the line of sight can be written as a an integral that includes a binary membership function for the object. (b) A single range scan, consisting of a number of measurements along lines of sight emanating from the point at the left, creates surface in 3D (shown here as a curve in the plane). A circular model deforms to fit the data (motion indicated by arrows).

where the variance σ_i^2 associated with a range reading r_i is represented as a confidence measure $c_i = 1/\sigma_i^2$. The term $k(c_i)$ can be ignored, because it does not depend on surface position and will not affect the outcome of the optimization. The squared distance of the surface to the range measurement can be represented as an integral

$$(s_i - r_i)^2 = \int_0^{r_{\max}} (\alpha - r_i)\, I_s(\alpha) \mathrm{d}\alpha - \int_{r_i}^{r_{\max}} (\alpha - r_i)\, \mathrm{d}\alpha \tag{7}$$

where $I_s(\alpha)$ is a function that is unity inside the object and zero outside, and $r_{\max}$ indicates the maximum effective range of the range finder. The term $r_{\max}$ serves as a bound on the error if the line of sight fails to intersect the surface, and it also serves to limit the range of influence of a particular range reading. That is, surface points that lie beyond $r_{\max}$ have no impact on the conditional likelihood associated with that range measurement.

Strictly speaking, this formulation is valid only if the surface intersects the line of sight at most once between the scanner and $r_{\max}$, as shown in figure 1(a). Thus, the segment along the line of sight with length $r_{\max}$ should be long enough to penetrate the model (as it deforms) but not so long that it emerges from the other side or enters the object again, as could easily happen with self occlusions. Typically, $r_{\max}$ should depend on r_i, e.g. $r_{\max} = r_i + \varepsilon$. The use of $r_{\max}$ and ε serves to limit the effects of the range data and thereby simplifies the mathematical model by ignoring self occlusions of the object. Unfortunately, in order for this to work properly, ε must be chosen so that scans to do not interfere with each other in areas where the object self occludes. This is not always possible, but with modifications to the minimization algorithm, discussed in successive sections, this problem can be all but eliminated.

Instead of putting bound $r_{\max}$ on the integral, we can use a *windowing function*, ω, that nullifies the effects of surface that lie beyond $r_{\max}$, i.e.,

$$\omega(r_{\max}, \alpha) = \begin{cases} 1 & \text{if } r_{\max} - \alpha \geq 0 \\ 0 & \text{if } r_{\max} - \alpha < 0 \end{cases} \tag{8}$$

The error becomes

$$(s_i - r_i)^2 = \int_0^{\infty} (\alpha - r_i)\, I_s(\alpha) \omega(r_i + \varepsilon, \alpha) \mathrm{d}\alpha - \int_{r_i}^{\infty} (\alpha - r_i)\, \omega(r_i + \varepsilon, \alpha) \mathrm{d}\alpha. \tag{9}$$

Given this formulation, the windowing function $\omega(\cdot)$ need not be binary; it could implement a fuzzy cutoff of the influence of the range data. Results in later sections use a Gaussian centered at the range reading with standard deviation ϵ.

To extend this formulation to include all of the samples in a single scan, R, we represent the distance along the line of sight in 3D. Let $\mathbf{n}_i$ be the unit vector along the line of sight and $\mathbf{r}_i = r_i \mathbf{n}_i$ be the 3D location of the ith range reading. Then

$$\begin{aligned} -\ln\left(P(R|\mathcal{S})\right) &= -\ln\left(P(\mathbf{r}_1, \ldots, \mathbf{r}_n|\mathcal{S})\right) \\ &= \sum_i c_i \int_0^\infty \left((\alpha\mathbf{n}_i - \mathbf{r}_i)\cdot\mathbf{n}_i\right) \omega\left((\alpha\mathbf{n}_i - \mathbf{r}_i)\cdot\mathbf{n}_i\right)\ I_{\mathcal{S}}(\alpha\mathbf{n}_i)\mathrm{d}\alpha - k \\ &= \sum_{u,v} c_{u,v} \int_0^\infty \left((\alpha\mathbf{n}_{u,v} - \mathbf{r}_{u,v})\cdot\mathbf{n}_{u,v}\right) \omega\left((\alpha\mathbf{n}_{u,v} - \mathbf{r}_{u,v})\cdot\mathbf{n}_{u,v}\right)\ I_{\mathcal{S}}(\alpha\mathbf{n}_{u,v})\mathrm{d}\alpha - k \end{aligned} \tag{10}$$

where $I_{\mathcal{S}} : \mathbb{R}^3 \mapsto \mathbb{R}$ is unity inside the object and zero otherwise, and

$$k = \sum_{u,v} c_{u,v} \int_{r_{u,v}}^\infty \omega\left((\alpha\mathbf{n}_{u,v} - \mathbf{r}_{u,v})\cdot\mathbf{n}_{u,v}\right)\ (\alpha\mathbf{n}_{u,v} - \mathbf{r}_{u,v})\cdot\mathbf{n}_{u,v}\ \mathrm{d}\alpha, \tag{11}$$

which does not depend on the surface. The notation u, v refers to the fact that the measurements from a single range map are arranged in a 2D grid. If we assume that this grid is relatively dense we can approximate the likelihood of the jth scan as an integral:

$$\begin{aligned} -\ln\left(P(R^{(j)}|\mathcal{S})\right) \approx \int\int\int & c(\varphi,\theta)\left((\alpha\mathbf{n}(\varphi,\theta) - \mathbf{r}(\varphi,\theta))\cdot\mathbf{n}(\varphi,\theta)\right) \\ & \omega\left((\alpha\mathbf{n}(\varphi,\theta) - \mathbf{r}(\varphi,\theta))\cdot\mathbf{n}(\varphi,\theta)\right)\ I_{\mathcal{S}}(\alpha\mathbf{n}(\varphi,\theta))\ \mathrm{d}\alpha\ \mathrm{d}\varphi\ \mathrm{d}\theta - k, \end{aligned} \tag{12}$$

where φ and θ are the continuous versions of u and v, respectively, and $\mathbf{n}(\cdot)$, $\mathbf{r}(\cdot)$, $c(\cdot)$, are continuous functions derived from some suitable interpolation (e.g., bilinear) of their discrete counterparts.

Because the rays that define the lines of sight associated with a single scan do not cross, there is unique mapping from each point within the 3D subspace swept out by the range finder to the point (φ, θ) within the range map that has a line of sight passing through that 3D point. Therefore, the volume integral of equation (13) can be reformulated in Cartesian coordinates:

$$-\ln\left(P(R^{(j)}|\mathcal{S})\right) \approx \int_{\mathcal{R}} c(\mathbf{x})\ D(\mathbf{x})\ \omega\left(D(\mathbf{x})\right)\ I_{\mathcal{S}}(\mathbf{x})\gamma(\mathbf{x})\ \mathrm{d}\mathbf{x}\ - k, \tag{13}$$

where $\gamma(\mathbf{x})$ is an integration factor, such that $\gamma(\mathbf{x})\mathrm{d}\mathbf{x} = \mathrm{d}\varphi\mathrm{d}\theta\mathrm{d}\alpha$. The term $D(\mathbf{x}) = \left((\mathbf{x} - \mathbf{r}(\mathbf{x}))\cdot\mathbf{n}(\mathbf{x})\right)$ is the signed distance from the surface position to the range measurement, and $\mathcal{R}$ is the volume swept out by the range finder. If we define the confidence $c(\mathbf{x})$ to be zero for any point outside the space swept out by the scanner, then one can extend bounds of the integral to $\mathbb{R}^3$. Because, $I_{\mathcal{S}}$ is a binary function that is one only within the object, the integral can be re-expressed over the object itself

$$-\ln\left(P(R^{(j)}|\mathcal{S})\right) \approx \int_{\Omega} c(\mathbf{x})\ D(\mathbf{x})\ \omega\left(D(\mathbf{x})\right)\ \gamma(\mathbf{x})\ \mathrm{d}\mathbf{x}\ - k. \tag{14}$$

The gradient descent on the posterior for a single range map is therefore

$$\frac{\partial\mathcal{S}}{\partial t} = -c(\mathcal{S})\ D(\mathcal{S})\ \omega\left(D(\mathcal{S})\right)\ \gamma(\mathcal{S})\ \mathcal{N}, \tag{15}$$

where $\mathcal{N}$ is the surface normal.

Equation (15) describes the motion of a model as it seeks to maximize its likelihood of giving rise to a single range map R, which is set of range measurements. A single range map creates a surface in 3D (shown in figure 1(b) as a curve in the plane), and the model expands or contracts, with a magnitude proportional to the distance along the line of sight to the range reading, so that it exactly fills the volume behind that surface. The windowing function $\omega(\cdot)$ controls the depth of the region behind the range surface over which this expanding or contracting action occurs. The

function $\gamma(\cdot)$ describes the unit-volume relationship between the scanner coordinates and cartesian coordinates. For the results in this paper we assume that the rays (lines of sight) emanating from the scanner are nearly parallel and regularly sampled, so that $\gamma(\cdot)$ is a constant.

The effect of multiple range maps is additive, and the Euler-Lagrange of the combined posterior is

$$\frac{\partial \mathcal{S}}{\partial t} = -g(\mathcal{S})\mathcal{N} + \rho(\mathcal{S}), \tag{16}$$

where

$$g(\mathbf{x}) = \sum_j c^{(j)}(\mathbf{x})\ D^{(j)}(\mathbf{x})\ \omega\left(D^{(i)}(\mathbf{x})\right)\ \gamma^{(j)}(\mathbf{x}), \tag{17}$$

and the superscripts indicate the range, line-of-sight, and confidence functions associated with a particular range map. The term $\rho(\mathcal{S}) = -\delta \ln P(\mathcal{S})$ is the Euler-Lagrange of the prior. In the absence of any prior (i.e., uniform prior), $\rho(\mathcal{S}) = 0$, and maximizing the combined effects of a sequence of range maps is linear; the solution is given by the zero crossings of the 3D function defined in equation (17). Indeed, the algorithms proposed by (Curless and Levoy 1996), as well as (Hoppe et al. 1992) and (Hilton et al. 1996), can be formulated as a special cases of this MAP approach, with a particular choice of $\omega(\cdot)$ and $c(\cdot)$, and with a uniform prior.

There are several reasons for going to an iterative scheme for finding optimal solutions. First is the use of a prior. In surface reconstruction, even a very low level of noise can degrade the quality of the rendered surfaces in the final result, and in such cases better reconstructions can be obtained by introducing a prior. Second is aliasing. Discretizing $g(\mathbf{x})$ and finding the zero crossings will cause aliasing in those places where the transition from positive to negative is particularly steep. A deformable model can place the surface much more precisely. The third reason for going to an iterative scheme is that despite the windowing function $\omega(\mathbf{x})$ there is interference between different range maps at places of high curvature. This problem is addressed by introducing a nonlinearity which is solved in an iterative scheme as described in Section 6.1. In this work, solutions of the linear problem will serve as the initial conditions for the nonlinear, iterative optimization strategy that results from the inclusion of a prior and a nonlinear term that compensates for lack of any explicit model of self occlusions.

3.1 The Prior

There are numerous possibilities for selecting priors that are consistent with the MAP formulation of the previous section. These possibilities range from high-level priors that bias the solution to look like certain shapes (Johnson 1993) to low-level priors that enforce some very general properties on those shapes. The goal of this work is to obtain a somewhat general reconstruction algorithm that can be subsequently tuned to specific applications. Therefore, we use a low-level prior that biases solutions toward smooth, continuous surfaces, like those associated with many man-made objects or anatomical structures. Of course, the selection of a prior will depend strongly on the application; the choice of prior for the reconstruction of injection-molded, hand-held objects should probably differ from the prior needed to reconstruct the shapes of highly irregular objects, such as the human brain with its many folds and protrusions.

For many applications, a natural choice of prior is to penalize the integral of the normalized first derivatives on the object surface, i.e., penalize surface area. This choice is based on the heuristic that many physical processes, both synthetic and natural, tend to conserve surface area and produce objects that reflect, at some level of detail, this tendency to minimize area. Alternatively, one could say that given a set of surfaces that are near the data, the algorithm should choose a surface that has less area. Often, but not always, this will be the smoother surface. A probability distribution for the prior that reflects this principle is

$$P(\mathcal{S}) = \left(\frac{\beta}{\Pi}\right)^{\frac{1}{2}} \exp\left(-\beta \int_{\mathcal{S}} \mathcal{N} \cdot \mathcal{N} \mathrm{d}\mathbf{x}\right) \tag{18}$$

After the logarithm, the Euler-Lagrange of this quantity is the mean curvature of the surface:

$$\rho(\mathcal{S}(r,s,t)) = \beta H(\mathcal{S}) = \beta \left(\frac{\partial^2 \mathcal{S}}{\partial r^2} + \frac{\partial^2 \mathcal{S}}{\partial s^2} \right) \tag{19}$$

where r, s is a local, orthonormal parameterization.

The term β controls the probability distribution of surfaces in the prior. A larger β means that the prior favors more strongly surfaces that are smoother. The parameter β could be set in any number of different ways. One way would be to "calibrate" the system by scanning a number of known objects and determine which values of β give the most accurate/reliable reconstructions. Another method would be to take a collection of objects that represent the application domain, model them, and generate a probability density function for surface area, possibly at different scales or resolutions. For this work, we treat β as a free parameter that must be tuned by the person using the algorithm.

The existence of a free parameter β represents a typical property of signal processing systems that must deal with noise in the absence of specific information about the signal. Results in the following sections will show that the algorithm is somewhat robust with respect to this free parameter, β, and that the algorithms fails gradually as this parameter is set too high or low. This paper will also show that very small values of β can reduce the effects of uncorrelated noise without distorting the overall shapes of the objects being reconstructed.

Despite the usefulness of this second-order smoothing term, it is somewhat limited in its effectiveness because it can interpolate only position and tends to create straight lines, flat surfaces, or singularities where there is little data. The topic of developing more effective low-level priors (Whitaker 1995), as well as incorporating high-level priors, is an area of ongoing investigation. Fourth-order terms, for example, could create structures with smoothly varying normals and allow the MAP approach to operate with less data (Sethian 1996). This paper will show that despite the limitations of the second-order smoothing given in equation (19), that prior is an improvement over none at all, especially in cases of noisy surface data.

4 Level-Set Models

The hill-climbing optimization strategy described in the previous section requires a modeling technology that is capable of accommodating incremental changes surface shape in an efficient manner. The equation of motion given by (16) makes no assumptions about the type surface model used to achieve the reconstruction. The MAP formulation could be adapted to any number of conventional, parametric surface models by expressing changes in surface position in terms of the parameters that control surface shape.

However, there are some drawbacks to using parametric deformable models. For instance, as models evolve and undergo large deviations from their original shapes, surface parameterizations often introduce de facto constraints. The expansion of polygonal models can create a kind of coarseness which prevents the model from capturing smaller structures; thus the evolution of polygonal models requires the creation and deletion of polygons (Miller et al. 1991, MacDonald, Avis and Evans 1994, Chen and Médioni 1994), or alternatively, a reparameterization (McInerny and Terzopoulos 1995, DeCarlo and Metaxas 1995). Also the choice a surface model with relatively few degrees of freedom, such as superquadric or superelipsoid, restricts solutions to the relatively small space of shapes that are captured by those modeling technologies.

An alternative to a parametric model, is a level-set model (Sethian 1996), i.e., a model that treats a 3D surface as the level-set of a discretely-sampled volume. Previous work has shown promising results with this modeling technology for 3D reconstruction, primarily in the context of 3D medical data (Malladi et al. 1995, Whitaker and Chen 1994, Whitaker 1994). This section gives the formulation for this modeling strategy, starting with an equation of motion for a deformable surface (Whitaker and Chen 1994). This formulation differs somewhat from that of (Caselles et al. 1995), which they develop by applying a conformal mapping to the energy function defined over the range.

The strategy is to rely on the properties of regular surfaces, which have local parameterizations that can be ex-

pressed in terms of intrinsic quantities such as arc length, curvature, etc. If one starts with a parameterized surface and the associated equations of motion, and then removes the parameterization from the deformable model, all that remains is intrinsic geometry, which is expressed in terms of the embedding, ϕ. The strategy for embedding active surfaces consists of four steps:

1. Express the equations of motion for a deformable model in terms of some unspecified parameterization (as was done in Section 3).
2. Describe the parameterization in terms of the differential structure of the model.
3. Assume the model is the level set of a function ϕ.
4. Express the geometry of the level set in terms of the differential structure of ϕ and create an evolution equation for ϕ.

In order to apply this strategy to a deformable surface $\mathcal{S}$, represent $\mathcal{S}$ as a level set of an 3D scalar function, $\phi : \mathbb{R}^3 \times \mathbb{R}^+ \mapsto \mathbb{R}$, which evolves over time. The evolution equations of the individual level surfaces imply corresponding evolution equations for the scalar function $\phi(\mathbf{x}, t)$, where $\mathbf{x} \in \mathbb{R}^3$.

A parametric deformable model, $\mathcal{S}(r, s, t)$ where $r, s \in U \subset \mathbb{R}$, can be represented

$$\mathcal{S} = \{\mathbf{x} | \phi(\mathbf{x}, t) = k\}. \tag{20}$$

The surface $\mathcal{S}$ remains a level set of ϕ over time, and therefore the time derivative is zero:

$$\frac{\partial \phi(\mathcal{S}, t)}{\partial t} + \nabla \phi(\mathcal{S}, t) \cdot \frac{\partial \mathcal{S}}{\partial t} = 0, \tag{21}$$

where

$$\nabla \phi(\mathbf{x}) \triangleq \left(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial z} \right). \tag{22}$$

Thus,

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\partial \mathcal{S}}{\partial t} = |\nabla \phi| \frac{\partial \mathcal{S}}{\partial t} \cdot \mathcal{N}, \tag{23}$$

where $\mathcal{N} = -\nabla\phi / |\nabla\phi|$ is the surface normal.

The data term for reconstruction from equation (16) takes the form

$$\frac{\partial \mathcal{S}}{\partial t} = -g(\mathbf{x}) \mathcal{N} \tag{24}$$

where $g(\cdot)$ is the cumulative effect from all of the individual range maps as in equation (17). The level-set formulation, without the prior, becomes

$$\frac{\partial \phi}{\partial t} = g(\mathbf{x}) |\nabla \phi| \tag{25}$$

The mean curvature, used for the prior, from equation (19) is computed directly from the first- and second-order structure of ϕ,

$$\begin{aligned} |\nabla \phi| \ (\mathcal{S}_{rr} + \mathcal{S}_{ss}) \cdot \mathcal{N} \quad = \quad & \left(\phi_x^2 + \phi_y^2 + \phi_z^2\right)^{-\frac{1}{2}} \left[\left(\phi_y^2 + \phi_z^2\right) \phi_{xx} + \left(\phi_z^2 + \phi_x^2\right) \phi_{yy} \right. \\ & \left. + \left(\phi_x^2 + \phi_y^2\right) \phi_{zz} - 2\phi_x \phi_y \phi_{xy} - 2\phi_y \phi_z \phi_{yz} - 2\phi_z \phi_x \phi_{zx} \right]. \end{aligned} \tag{26}$$

In the numerical implementation, the derivatives are calculated using centralized differences (Sethian 1996).

Combining the data term and the prior gives the following level-set formulation:

$$\begin{aligned} \frac{\partial \phi}{\partial t} \quad = \quad & g(\mathbf{x}) |\nabla \phi| + \beta \left(\phi_x^2 + \phi_y^2 + \phi_z^2\right)^{-\frac{1}{2}} \left[\left(\phi_y^2 + \phi_z^2\right) \phi_{xx} + \left(\phi_z^2 + \phi_x^2\right) \phi_{yy} \right. \\ & \left. + \left(\phi_x^2 + \phi_y^2\right) \phi_{zz} - 2\phi_x \phi_y \phi_{xy} - 2\phi_y \phi_z \phi_{yz} - 2\phi_z \phi_x \phi_{zx} \right] \end{aligned} \tag{27}$$

Equation (27) is invariant under certain kinds of geometric transformations. First, it is invariant to orthogonal group transformations on $\mathbf{x}$. Thus the position and orientation of the model or the data has no impact (to within the

error introduced by the representation of ϕ) on the solution. Equation (27) is also invariant to monotonic transformations on ϕ; that is, equation (27) acts only on level sets and treats each level set of ϕ as an individual surface evolving under the same set of set of priors and forces as all other level sets of ϕ.

4.1 Numerical Algorithms

The differential equation described by equation (27) has two parts. The first part is a first-order term that has the form $g(\mathbf{x})|\nabla\phi(\mathbf{x})|$. It is a moving wave front with a velocity that depends on position. This expression cannot be solved with a simple forward finite difference scheme. Such schemes tend to overshoot, and they are unstable. To solve this problem Osher and Sethian (1988) propose an *up-wind* scheme. The up-wind method uses a one-sided derivative that looks in the up-wind direction of the moving wavefront, and thereby avoids the overshooting associated with forward finite differences.

The one-sided derivatives are denoted by $\mathrm{d}^{(+)}$ and $\mathrm{d}^{(-)}$. Let $u_{x,y,z}$ with domain X be an approximation to ϕ defined on a discrete rectilinear grid with spacing h. The one-sided derivatives are

$$\mathrm{d}_x^{(+)} u_{x,y,z} \triangleq (u_{x+h,y,z} - u_{x,y,z})/h, \tag{28}$$

$$\mathrm{d}_x^{(-)} u_{x,y,z} \triangleq (u_{x,y,z} - u_{x-h,y,z})/h. \tag{29}$$

For the first term of equation (27) the up-wind scheme has the following form (Osher and Sethian 1988):

$$\begin{aligned} g(x,y,z)|\nabla u| = \Big(& \left[\max\left(g(x,y,z),0\right)\ \mathrm{d}_x^{(+)}u + \min\left(g(x,y,z),0\right)\ \mathrm{d}_x^{(-)}u\right]^2 \\ & + \left[\max\left(g(x,y,z),0\right)\ \mathrm{d}_y^{(+)}u + \min\left(g(x,y,z),0\right)\ \mathrm{d}_y^{(-)}u\right]^2 \\ & + \left[\max\left(g(x,y,z),0\right)\ \mathrm{d}_z^{(+)}u + \min\left(g(x,y,z),0\right)\ \mathrm{d}_z^{(-)}u\right]^2 \Big)^{\frac{1}{2}}, \end{aligned} \tag{30}$$

where the time steps are limited by the speed of the fastest moving wavefront,

$$\Delta t \leq \frac{1}{\sup_{x,y,z\in X}\{|\nabla g(x,y,z,t)|\}} \tag{31}$$

The second term in equation (27) is a diffusion term that can be solved using a finite forward difference scheme and does not require the up-wind method.

4.2 Narrow-Band Methods

If one is interested in only a single level set, the formulation described previously is not computationally efficient. This is because solutions are usually computed over the entire domain of u. The solutions, $u_{x,y,z}(t)$ describe the evolution of an embedded family of contours. While this dense family of solutions might be advantageous for certain applications, there are other applications that require only a single surface model. In such applications the calculation of solutions over a dense field is an unnecessary computational burden, and the presence of contour families can be a nuisance because further processing might be required to extract the level set that is of interest.

When solving for only a single level set, $\phi(\mathbf{x},t) = k$, the evolution of ϕ is important only in the vicinity of that level set. The evolution of the implicit models is such that the level sets evolve independently (to within the error introduced by the discrete grid). Thus, one should perform calculations for the evolution of ϕ only in a neighborhood of the surface $\mathcal{S} = \{\mathbf{x}|\phi(\mathbf{x}) = k\}$. In the discrete setting, there is a particular subset of grid points whose values control a particular level set (see figure 2).

Adalstein and Sethian (1995) propose a *narrow-band* approach that takes advantage of the fact that the movement of a particular level set is a local phenomenon. The narrow-band technique constructs an embedding of the evolving curve or surface via a signed distance transform. The distance transform is computed over a finite width of only m points, and the remaining points are set to constant values to indicate that they do not lie within the narrow band, or *tube* as they call it. The evolution of the surface (they demonstrate it for curves in the plane) is computed by

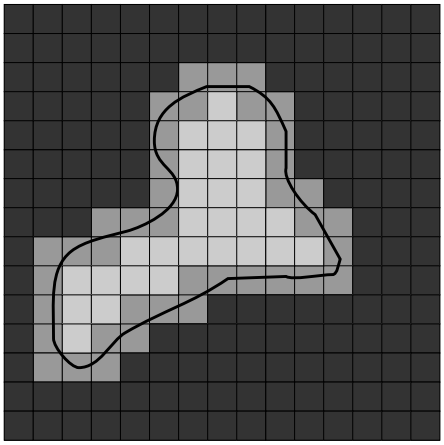


Figure 2: A level curve of a 2D scalar field passes through a finite set of cells. Only those grid points nearest to the level curve are relevant to the evolution of that curve.

calculating the evolution of u only on the set of grid points that are within a fixed distance to the initial level set, i.e. within the narrow band. When the evolving level set approaches the edge of the band, they calculate a new distance transform and a new embedding, and they repeat the process. This algorithm relies on the fact that the embedding is not a critical aspect of the evolution of the level set. That is, the embedding can be transformed or recomputed at any point in time, so long as such a transformation does not change the position of the kth level set, and the evolution will be unaffected by this change in the embedding.

Despite the improvements in computation time, the narrow-band approach is not optimal for several reasons. First it requires a band of significant width ($m = 12$ in the examples of Adalstein and Sethian (1995)) where one would like to have a band that is only as wide as necessary to calculate the derivatives of u near the level set (e.g. $m = 2$). The wider band is necessary because the narrow-band algorithm trades off two competing computational costs. One is the cost of stopping the evolution and computing the position of the curve and distance transform (to sub-cell accuracy) and determining the domain of the band. The other is the cost of computing the evolution process over the entire band. The narrow-band method also requires additional techniques, such as smoothing, to maintain the stability at the boundaries of the band, where some grid points are undergoing the evolution and nearby neighbors are stationary.

5 Sparse-Field Solutions

The narrow-band algorithm reduces computation time by restricting the updates to a *band* of grid points that lie near the level set. However, that strategy is based on the assumption that computing the distance transform is so costly that it cannot be done at every iteration of the evolution process — the band of computation must be wide enough to justify this costly update of the embedding.

The sparse-field algorithm proposed in this section uses an approximation to the distance transform that makes it feasible to recompute the neighborhood of the level-set model at each time step. Thus, it takes the narrow-band strategy to the extreme; it computes updates on a band of grid points that is only one point wide. The values of the points in the active set can be updated using the up-wind scheme and the mean-curvature flow described in the previous sections. When computing updates on so few points, however, one must be careful to maintain a neighborhood around those points so that the derivatives that control the process can be computed with sufficient accuracy. The strategy is to extend the embedding from the active points outward in layers to create a neighborhood around those points that is precisely the width needed to calculate the derivatives for the next time step.

This approach has several advantages. The algorithm does precisely the number of calculations needed to compute

the next position of the level curve. It does not require explicitly recalculating the positions of level sets and their distance transforms. For large 3D data sets, the very process of incrementing a counter and checking the status of all of the grid points is prohibitive. In the sparse-field algorithm the number of points being computed is so small, it is feasible to use a linked-list to keep track of them. Thus, at each iteration the algorithm *visits* only those points adjacent to the k-level curve. Also, the sparse-field approach identifies a single level set with a specific set of points whose values control the position of that level set. This allows one to compute external forces to an accuracy that is better than the grid spacing of the model, resulting in a modeling system that is more accurate for 3D reconstruction.

The k-level surface, S, of a function u defined on a discrete grid has a set of cells through which it passes, as shown in figure 2. The set of grid points adjacent to the level set is called the *active set*, and the individual elements of this set are called *active points*. As the model deforms the active will change. All of the derivatives (up to second order) required to calculate the update of u are computed using nearest neighbor differences. Therefore, only the active points and their neighbors are relevant to the evolution of the level-set at any particular time in the evolution process.

One important aspect of the sparse-field algorithm is the mechanism to control membership in the active set. In order to maintain stability, one must update the active set and neighboring points in a way that allows grid points to enter and leave the active set without those changes in status affecting their values. This mechanism can be understood as follows. Active points must be adjacent to the level-set model. Therefore their positions lie within a fixed distance to the model. Because the embedding is a distance transform, the values of u for locations in the active set must lie within a certain range of values. When active-point values move out of this *active range* they are no longer adjacent to the model. They must be removed from the set and other grid points, those whose values are moving into the active range, must be added to take their place. The precise ordering and execution of these operations is critical to the proper operation of the algorithm.

If we assume that the embedding u is a discrete approximation to the distance transform of the model, then the distance of a particular grid point, x_j, to the level set is given by the value of u at that grid point. If the distance between grid points is defined to be unity, then we should remove a point from the active set when the value of u at that point no longer lies in the interval $[-\frac{1}{2}, \frac{1}{2}]$ (see figure 3). If the neighbors of that point maintain their distance of 1, then those neighbors will move into the active range just as x_j is ready to be removed.

There are two operations that are significant to the evolution of the active set. First, the values of u at active points change from one iteration to the next. Second, as the values of active points move out of the active range they are removed from the active set and other, neighboring grid points are added to the active set to take their place. The appendix of this paper gives some formal definitions of active sets and the operations that affect them, and it shows that active sets will always form a boundary between positive and negative regions in the discrete sampling u, even as control of the level set passes from one set of active points to another.

Because grid points that are near the active set are kept at a fixed value difference from the active points, active points serve to control changes in the nonactive grid points to which they are adjacent. The neighborhoods of the active set are defined in *layers*, $L_{+1}, \ldots L_{+N}$ and $L_{-1}, \ldots L_{-N}$, where the i indicates the city-block distance from the nearest active grid point, and negative numbers are used for the outside layers. For notational convenience the active set is denoted L_0.

The number of layers should coincide with the size of the footprint or neighborhood used to calculate derivatives. In this way, the inside and outside grid points undergo no changes in their values that affect or distort the evolution of the zero set. The work in this paper uses first- and second-order derivatives computed on a a $3 \times 3 \times 3$ kernel (city-block distance 2 to the corners). Therefore only five layers are necessary: 2 inside layers, 2 outside layers, and the active set. These layers are denoted L_1, L_2, L_{-1}, L_{-2}, and L_0, respectively.

The active set has grid point values in the range $[-\frac{1}{2}, \frac{1}{2}]$. The values of the grid points in each neighborhood layer are kept 1 unit from the next layer closest to the active set (as in figure 3). Thus the values of layer L_i fall in the interval $[i - \frac{1}{2}, i + \frac{1}{2}]$. For $2N + 1$ layers, the values of the grid points that are not in any of the layers are either inside all of the layers, with a value of $N + \frac{1}{2}$, or outside all of the layers, with a value of $-N - \frac{1}{2}$. The procedure for updating the image and the active set based on surface movements is as follows:

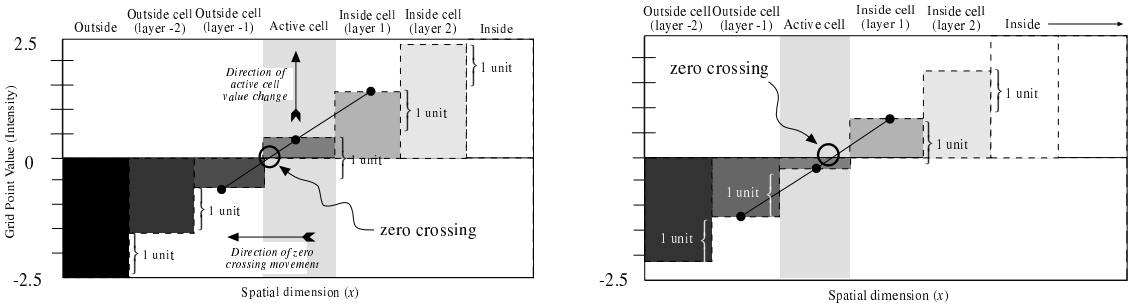


Figure 3: The status of grid points and their values at two different points in time show that as the zero crossing moves, *activity* is passed from one grid point to another.

1. For each active grid point, x_j, do the following:
   (a) Calculate the local geometry of the level set.
   (b) Compute the net change of u_{x_j}, based on the internal and external forces, using some stable (e.g., upwind) numerical scheme where necessary.
2. For each active grid point x_j add the change to the grid point value and decide if the new value $u_{x_j}(t + \Delta t)$ falls outside the $[-\frac{1}{2}, \frac{1}{2}]$ interval. If so, put x_j on lists of grid points that are changing status, called the *status list*; S_1 or S_{-1}, for $u_{x_j}(t + \Delta t) > \frac{1}{2}$ or $u_{x_j}(t + \Delta t) < -\frac{1}{2}$, respectively.
3. Visit the grid points in the layers L_i in the order $i = \pm 1, \ldots \pm N$, and update the grid point values based on the values (by adding or subtracting one unit) of the next inner layer, $L_{i \mp 1}$. If more than one $L_{i \mp 1}$ neighbor exists then use the neighbor that indicates a level curve closest to that grid point, i.e., use the maximum for the outside layers and minimum for the inside layers. If a grid point in layer L_i has no $L_{i \mp 1}$ neighbors, then it gets demoted to $L_{i \pm 1}$, the next level away from the active set.
4. For each status list $S_{\pm 1}, S_{\pm 2}, \ldots, S_{\pm N}$ do the following:
   (a) For each element x_j on the status list S_i, remove x_j from the list $L_{i \mp 1}$, and add it to the L_i list, or, in the case of $i = \pm(N + 1)$, remove it from all lists.
   (b) Add all $L_{i \mp 1}$ neighbors to the $S_{i \pm 1}$ list.

This algorithm can be implemented efficiently using linked-list data structures combined with arrays to store the values of the grid points and their states as shown in figure 4. This requires only those grid points whose values are changing, the active points and their neighbors, to be visited at each time step. Therefore computation time grows as m^{n-1}, where m is the number of grid points along one dimension of u. Computation time for dense-field approach increases as m^n. The m^{n-1} growth in computation time for the sparse-field models is consistent with conventional (parameterized) models, for which computation times increase with the resolution of the domain, rather than the range.

Another important aspect of the performance of the sparse-field algorithm is the larger time steps that are possible. In the numerical schemes for updating level-set models, the time steps are limited by the speed of the "fastest" moving level curve, i.e., the maximum of the force function. Because the sparse-field method calculates the movement of level sets over a subset of the image, time steps are bounded from below by those of the dense-field case, i.e.,

$$\sup_{x \in \mathcal{A} \subset X} (g(x)) \leq \sup_{x \in X} (g(x)), \tag{32}$$

where $g(x)$ is the space varying speed function and $\mathcal{A}$ is the active set.

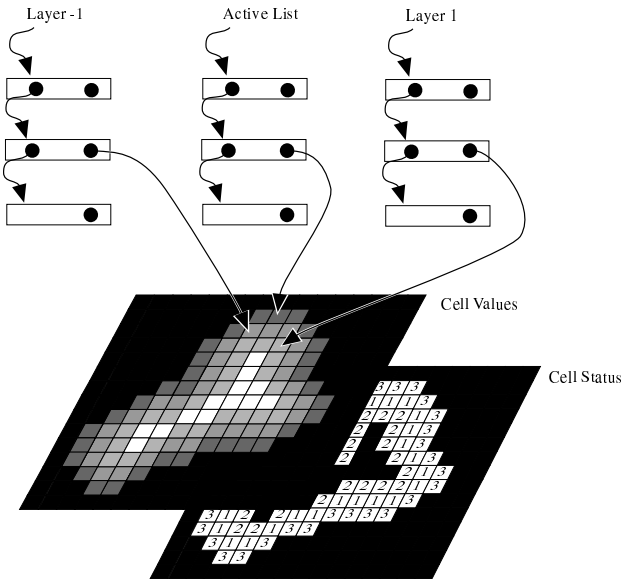


Figure 4: Linked-list data structures provide efficient access to those grid points with values and status that must be updated.

5.1 Empirical results of The Sparse-Field Algorithm

5.1.1 Error analysis

The sparse-field algorithm described above is based on an important approximation: grid points adjacent to the active points are assumed to undergo the same change in value as their nearby active-set neighbors. The dense-field algorithms treat each level set (and each grid point) separately. Because proximate level sets can have different shapes and undergo different forces, nearby grid points can undergo different changes in value. The question is how the sparse-field approximation affects the evolution of the zero-level set. The results in this section show that the evolution of the zero-level set in the sparse-field algorithm introduces an error that is consistent with that of the dense-field approach and that both errors are significantly smaller than the grid spacing h.

In order to measure the effects of these approximations we compare the movement of the level sets computed by the both the dense-field and sparse-field methods to the deformation of a circle, which can be computed analytically (Adalstein and Sethian 1995). The total error of a model is computed from the set of zero crossings along the lines connecting the grid. These points are found by using a linear interpolation between adjacent points that lie on either side of a zero crossing. The total error is the average squared distance of these zero crossings from the ideal.

Figure 5a shows the error of the dense- and sparse-field methods in 2D for a circle moving under its own curvature. Figure 5b shows the same results for the a circle moving in the direction of the inward normal at a uniform speed of 1. The 2D grid is 100×100, and the grid spacing is unity. The circle begins with a radius of 30 units. These results show that, on the whole, the sparse-field method and dense-field methods give comparable errors. The magnitude of these errors, when scaled appropriately for differences in time constants and grid spacing, are consistent with those documented in (Adalstein and Sethian 1995) even though the error metric is slightly different. Notice that in the case of constant inward velocity, both discrete level-set methods have high errors as the circle radius becomes quite small (it should be zero at $t = 30$). This is to be expected and is a inevitable consequence of using discrete methods to represent objects with sizes that are comparable to the grid spacing.

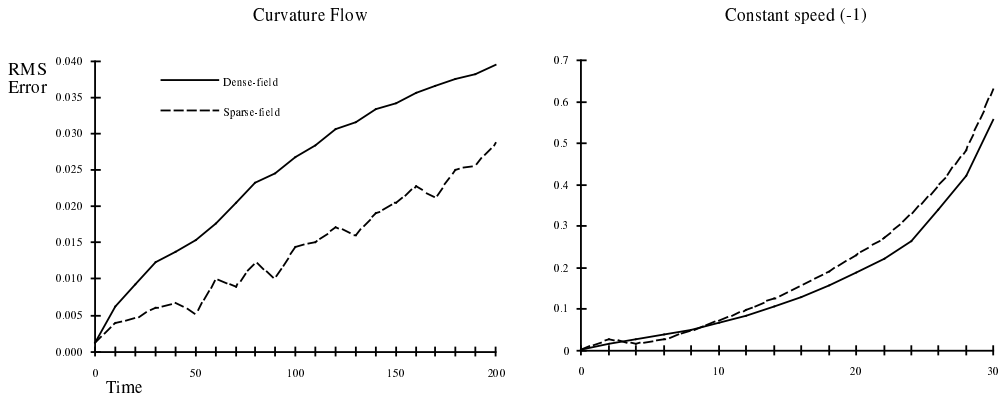


Figure 5: The rms errors, relative to ground truth, for a circle evolving under constant speed and curvature suggest that the errors associated with the sparse-field algorithm are no worse than those introduced by the discretization and level-set approximation.

5.1.2 Timing

Measurements of the timing bear out the expected improvements in performance with the sparse-field method. A rough calculation of the expected improvements is as follows. In 2D, an $n \times n$ image requires kn^2 calculations per update, where k is the number of calculations per grid point. The updates for the sparse field should be $k'n$, where k' includes the extra costs associated with updating and maintaining the neighborhood. Let $R = k/k'$, be the efficiency of the improvement. Of course, we expect R is less than one because of the extra overhead of maintaining the active set and the neighborhood around it. For a circle of radius $n/3$, the cardinality of the active set is approximately $2n$. The neighborhood is another 4 layers, each requiring 1 pass for an update. Let the cost of visiting maintaining the list and updating the neighborhood be denoted α. he Then the ratio of calculation times, R, for the two methods should be approximately

$$R = \frac{k}{k'} = \frac{k}{(4\alpha + 1)2k} = \frac{1}{8\alpha + 2}, \tag{33}$$

If α is of the same order magnitude as k (because it requires several floating point calculations involving nearest neighbors), then equation 33 gives some rough bounds on the efficiency: $0.012 \leq R \leq 0.36$.

Of course these numbers are estimates and depend quite heavily on the implementation. For the experiments of this section we have used a rather high-level, object-oriented, *C++* image processing library (developed in house), which uses in-line functions where practical as well as templated images and generic data structures. This library emphasizes ease of implementation rather than performance. The "inner loops" of the methods compared, i.e. those loops that do the calculations and updates at each pixel, are written with identical code where possible. All results are computed on a Sun Sparc 10.

Table 1 gives the average time per iteration, averaged over 25 iterations, for a circle of radius $n/3$ moving with first curvature and then a constant inward speed. The execution times and the efficiency factors confirm the expected improvements in performance from the sparse-field method.

Making direct comparisons of these computation times with the results of other researchers is difficult because of differences in implementations and hardware. However, the ratio of the dense-field computation time to that of the sparse-field algorithm gives a *performance ratio*. We have computed the same performance ratio for the narrow-band method from the data in (Adalstein and Sethian 1995). Figure 6 shows a graph of that ratio for the sparse-field approach and the narrow-band method for two distinct band widths, 6 and 12. These results show that the sparse-field method is somewhat faster, and the improvements in computation time grow as the domain size increases. For larger models, the difference in computation time between these methods is even more extreme. The models in these experiments are relatively small and the time difference should be more dramatic as one goes to 3D. For instance,

Image Width	Sparse-Field	Dense-Field	Efficiency Factor R
Circle moving under curvature.			
75	0.02	0.12	0.080
150	0.03	0.49	0.109
300	0.07	1.96	0.093
600	0.14	7.87	0.094
Circle moving at constant speed -1.			
75	0.02	0.11	0.073
150	0.05	0.44	0.059
300	0.10	1.74	0.058
600	0.20	6.97	0.058

Table 1: A comparison of execution times (seconds/iteration) for computing the evolution a circle

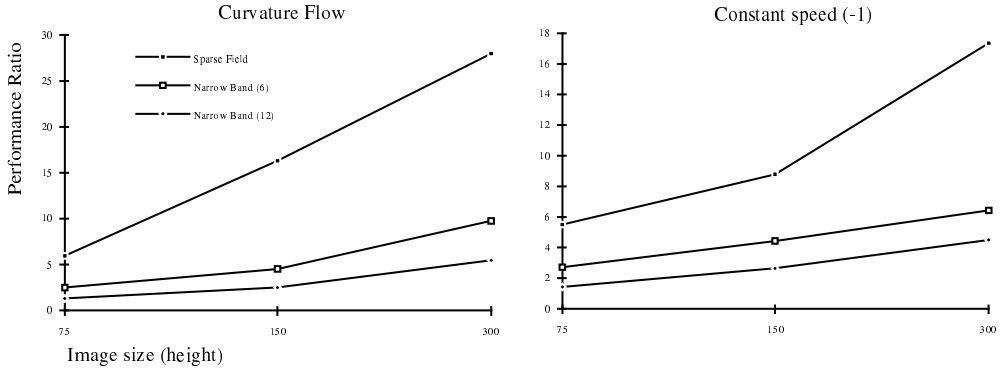


Figure 6: A comparison of the sparse-field method with the narrow-band results from (Adalstein and Sethian 1995) bears out the advantage of the sparse field method. The advantage is modest for small models but grows as the models get larger — an important trend when considering large, volumetric models.

(a) (b) (c)

Figure 7: The deformation of a cube moves downhill on the distance transform of a torus. In (a) the steady state of the dense-field approach shows that the torus suffers from aliasing due to the shocks which form near the boundary of the torus. In (b) the sparse-field algorithm bounces back and forth between grid points that have forces in opposite directions. In (c) the modified sparse-field algorithm uses a first-order approximation to position the zero-level set to subcell accuracy.

the circle in the 300×300 image contains approximately 600 pixels along its border. The surface of a sphere in a $300 \times 300 \times 300$ volume would contain about 40,000 voxels.

5.2 Subcell accuracy

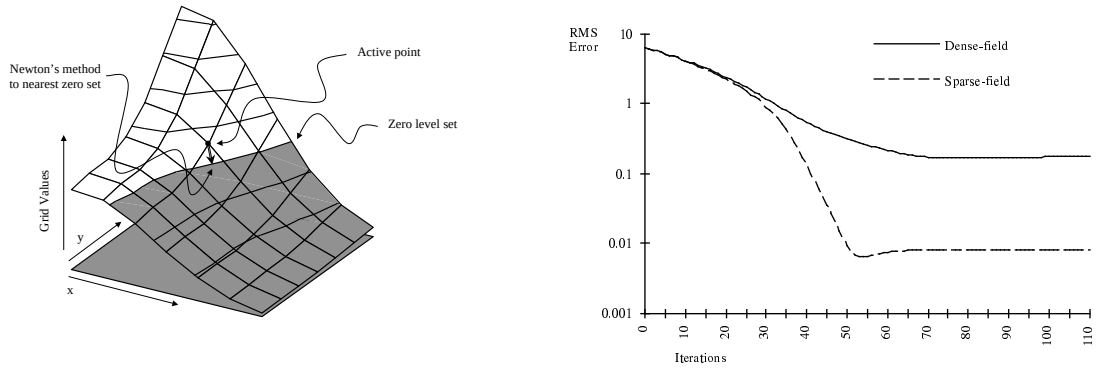
Results of the previous section demonstrate the accuracy and computational efficiency of the sparse-field algorithm. However, the level-set approach is still not appropriate for 3D reconstruction because of its limited spatial resolution. This section shows how the sparse-field algorithm can be modified to improve the overall accuracy of the level-set method.

It is well known that front propagations of the form of equation (25), without any smoothing term, form shocks where fronts moving in opposite directions meet. In the discrete domain, when using an upwind scheme, these shocks take the form of high contrast regions that form along those grid points that lie near the zero crossings of $g(\cdot)$. Unfortunately these shocks have an adverse side effect; they constrain the positions of level-set solutions to fall on the grid lines, half way between grid points. The high contrast regions associated with shock formation cause *aliasing* in the final results of level-set models.

This aliasing is not an inherent property of the implicit representation. Indeed, grid-point values can be manipulated to position zero crossings anywhere on the grid lines, and linear or higher order interpolation techniques can be used to construct parametric representations from level sets to within subcell accuracy. The aliasing associated with the moving wavefronts follows from the fact that the numerical schemes for propagating fronts sample the force function $g(\cdot)$ only at grid point locations; it is an inherent problem in the level-set numerical schemes that have been proposed to date. Any iterative deformation scheme that samples the forcing function at only a discrete set of grid locations will be limited in its ability to accurately locate the solution.

The problems associated with the discrete sampling of the forcing function are particularly troublesome when considering 3D reconstruction. Figure 7(a) shows the result of allowing a cube (sampled on a $50 \times 50 \times 50$ grid) to move on the distance transform of a torus. The distance transform of the torus (which serves as the $g(\mathbf{x})$ for the reconstruction) is computed analytically and then sampled on the same grid as the cube. The level-set model forms patterns that reflect the underlying grid structure because of the shocks that form between grid points that are inside and outside of the torus boundary. The problems of shock creation and aliasing are even more serious in cases where the forces, represented by $g(\cdot)$, have a greater resolution than the model — as in the case of recovering 3D models from high-resolution range maps. In such cases limiting the model position to the resolution of the grid is far from optimal; it introduces unnecessary artifacts.

The sparse-field algorithm provides a mechanism for positioning level sets to subcell accuracy. In the sparse-field algorithm the active grid points can be thought of as control points for a nearby zero-level set. The level set does not necessarily pass through the center of the grid point (except in the special case where the grid point has a value zero).



(a) (b)

Figure 8: The modified sparse-field algorithm: (a) For a curve in a plane, the position of the level set near to a grid point is found using Newton's method. (b) For a square being fit to a circle, the sparse-field method obtains subcell accuracy and almost perfect fit (less than a hundredth of a voxel error), while the dense-field scheme forms shocks and stabilizes at an error of 0.20 cells.

Newton's method gives a first-order approximation to the position of the nearby zero-level set, as in figure 8(a).

Let $\mathbf{x} \in \mathrm{I}^D$ be the position of a grid point, $\nabla u_{\mathbf{x}}$ be the vector of first derivatives of the model at some time t. The position of the closest zero-level set to the grid point $\hat{\mathbf{x}}$ is given by

$$\hat{\mathbf{x}} = \mathbf{x} - u_{\mathbf{x}} \frac{\nabla u_{\mathbf{x}}}{\nabla u_{\mathbf{x}} \cdot \nabla u_{\mathbf{x}}}, \tag{34}$$

except when $\nabla u_{\mathbf{x}} = 0$, which must be handled as a special case. Computing the forces on level-set locations away from the grid points is similar in philosophy to the method of *extending* the velocity fields of level sets described by Sethian (1996).

The numerical computation of $\nabla u_{\mathbf{x}}$ in equation (34) can proceed in one of several different ways. Although centralized differences are possible, the closest zero crossing can be found by finding the *steepest* one-sided derivative in each dimension. Define the function

$$\mathrm{MaxAbs}(a, b) \triangleq \begin{cases} a & |a| > |b| \\ b & |a| < |b| \\ \frac{a+b}{2} & |a| = |b| \end{cases} . \tag{35}$$

The direction of nearest zero crossing can be calculating by using

$$\nabla u_{\mathbf{x}} \triangleq \mathrm{MaxAbs}(\mathrm{d}_{x_1}^{(+)} u_{\mathbf{x}}, \mathrm{d}_{x_1}^{(+)} u_{\mathbf{x}}), \ldots, \mathrm{MaxAbs}(\mathrm{d}_{x_D}^{(+)} u_{\mathbf{x}}, \mathrm{d}_{x_D}^{(+)} u_{\mathbf{x}}). \tag{36}$$

Figure 7(c) shows that better results are obtained from the modified sparse-field method (first-order approximation to the level set location) than methods which sample force-field values only at grid point locations. The first-order approximation allows grid points to achieve grey-level values that reflect their distance from nearby features. This first-order improvement is essential in using the level-set paradigm for modeling 3D objects. The rendering of 3D models makes use of first-order derivatives of the volume data, which are sensitive to aliasing artifacts.

Figure 8(b) shows the error, using the same error measure described in section 5.1.1, for a square moving downhill on the distance transform of a circle. The error begins at about 7 units and slowly decreases as the model moves toward the circle. The difference in the sparse-field and dense-field methods demonstrates the subcell accuracy that is obtained with the first-order modifications. The dense-field method forms shocks, and the error stabilizes at about 0.20 cells. The dense-field scheme produces a binary image which is the inside-outside function for the circle. This result is representative of other numerical algorithms for level sets, such as that of (Adalstein and Sethian 1995),

which do not attempt to position level sets to sub-cell accuracy. The sparse-field method positions the zero-level set to subcell accuracy and eventually obtains very small errors in fitting the level set. The small dip in error at about 50 iterations is even more dramatic in the dense-field approach but is not easily visible on the logarithmic scale given in figure 8(b). This overshoot indicates that the model moves through the solution and slightly beyond before reaching a steady state.

This strategy of approximating the level-set position to sub-cell accuracy can generalized to higher orders. For instance, in two dimensions one can fit a second-order function to u in the neighborhood of the active grid point. Such higher order schemes are not pursued in this paper for two reasons. First, they would add considerably to the computational burden of the method. Second, the embedding of the level set is the distance transform implies that $|\phi| = 1$ almost everywhere, and therefore second derivative in the gradient direction is virtually 0 except at singularities (Bruce and Giblin 1986).

6 Level-Set Models for 3D Reconstruction

This section combines the level-set modeling technology from Sections 4 and 5 with the MAP reconstruction formulation of Section 3 to generate 3D reconstructions of objects from multiple range maps. The strategy is as follows. Construct a rather coarse volume that is the solution to the linear problem, i.e. the zero-level sets of $g(\cdot)$, without the prior. This volume serves as initialization for a level-set model which moves toward the data given by the range maps while undergoing a second-order flow to enforce the prior. After the rate of deformation slows to below some threshold, the resolution is increased, the volume resampled, and the process repeated. The coarse-to-fine strategy is intended to be a continuation method (as described by (Nielson 1997, Snyder, Han, Bilbro, Whitaker and Pizer 1995)) for both reducing computation and preventing the algorithm from converging on local minima.

There are several additional considerations which affect the performance of this algorithm. The first consideration is that the MAP formulation in Section 3 suffers from a problem; it ignores the nonlinearity resulting from the fact that the range scanner gives the depth reading for the *single closest surface point* along the line of sight. Therefore the solution given by the zero sets of $g(\cdot)$ could contain artifacts that result from surfaces interacting at occlusion boundaries. The use of $r_{\max}$ or the windowing function $\omega(\cdot)$ help alleviate this problem, but it is virtually impossible to stop this interaction in places where the object has high curvature near its occluding contour. An iterative scheme can eliminate interactions of surfaces near occluding contours by taking advantage of the knowledge that the objects are solids. That is, a surface cannot return a range reading if it is facing away from the scanner. If the surface normal is taken to be outward, then the dot product of the surface normal and the line of sight must be negative. The evolution equation, modified to include only those surface that face the scanner associated with a particular range map, is

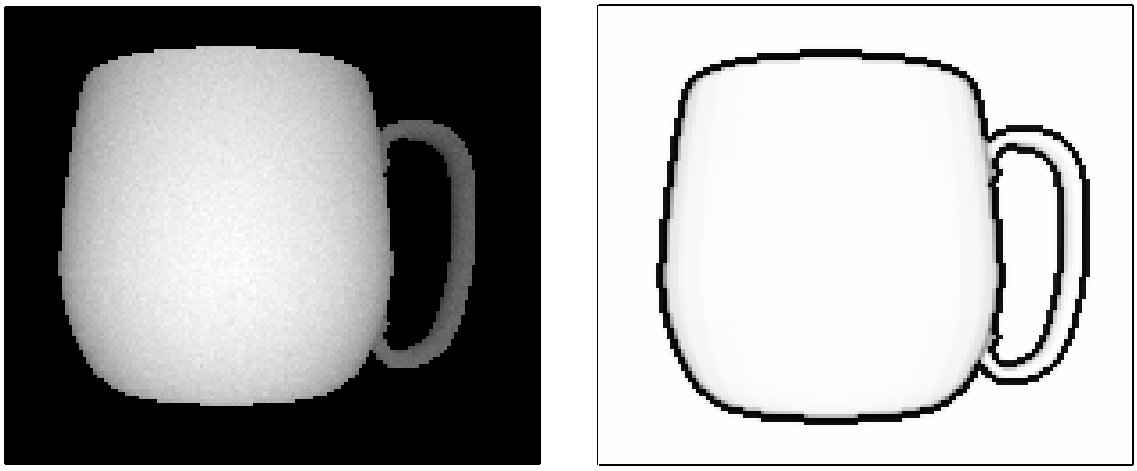
$$\frac{\partial \phi(\mathbf{x},t)}{\partial t} = |\nabla \phi(\mathbf{x})| \sum_j D^{(j)}(\hat{\mathbf{x}})\, \omega\left(D^{(i)}(\hat{\mathbf{x}})\right)\, \gamma^{(j)}(\hat{\mathbf{x}})\, c^{(j)}(\hat{\mathbf{x}}) \frac{\left(\nabla\phi \cdot \mathbf{n}^{(j)}(\hat{\mathbf{x}})\right)^+}{\nabla\phi \cdot \mathbf{n}^{(j)}(\hat{\mathbf{x}})} + \tag{37}$$

$$\begin{aligned} &\beta\left(\phi_x^2+\phi_y^2+\phi_z^2\right)^{-\frac{1}{2}}\left[\left(\phi_y^2+\phi_z^2\right)\phi_{xx}+\left(\phi_z^2+\phi_x^2\right)\phi_{yy}\right. \\ &\left.+\left(\phi_x^2+\phi_y^2\right)\phi_{zz}-2\left(\phi_x\phi_y\phi_{xy}+\phi_y\phi_z\phi_{yz}+\phi_z\phi_x\phi_{zx}\right)\right], \end{aligned} \tag{38}$$

where $\mathbf{n}^{(j)}(\mathbf{x})$ is the line of sight from a range finder to a 3D point, $\mathbf{x}$.

A second practical consideration is the confidence associated with the range data. In Section 3 the confidence measure $c^{(j)}(\mathbf{x})$ was expressed as the inverse of the variance of the sensor noise associated with the range measurement along the line of sight $\mathbf{n}(\hat{\mathbf{x}})$. Other factors also affect this confidence metric. One such factor is the uncertainty in the position of the model that results from the discretization of the embedding ϕ. More specifically, each grid point being updated in the volume u, controls the movement a of a level surface nearby, whose position is known to only finite accuracy. Thus there is cloud of uncertainty around each grid point. The first-order approximation to the level-set position described in Section 5.2 improves matters considerably, but the 3D positional error cannot be discounted entirely. For this work we assume that uncertainty is isotropic with variance denoted $\sigma_{\mathbf{x}}^2$.

The positional error associated with the discrete sampling of a volume gives rise to an uncertainty about the line of



(a) (b)

Figure 9: The range map of a mug, viewed as an image (a), gives rise to a confidence measure (b) that combines both the noise of the scanner with the spatial uncertainty of the model.

sight and thereby uncertainty about the value of the associated range measurement. Using a first-order approximation to calculate this error we have

$$c^{(j)}(\mathbf{x}) = \frac{1}{\left(\sigma^{(j)}(\mathbf{x})\right)^2 + \left|\nabla D^{(j)}(\mathbf{x})\right|^2 \sigma_{\mathbf{x}}^2(\mathbf{x})}, \tag{39}$$

where $\sigma^{(j)}(\mathbf{x})^2$ is the variance associated with the range measurement along the line of sight that passes through $\mathbf{x}$ and $D^{(j)}(\mathbf{x})$, as in equation (13), is the signed distance along the line of sight between the range measurement and $\mathbf{x} \in \mathbb{R}^3$. Derivatives of $D^{(j)}(\mathbf{x})$ combine the geometry of the scanner with derivatives of the range map. That is,

$$\nabla D^{(j)}(\mathbf{x}) = \nabla \mathbf{x} - \nabla r^{(j)}(\mathbf{x}) = \nabla \mathbf{x} - \frac{\partial r^{(j)}(\theta,\varphi)}{\partial \theta}\nabla\theta(\mathbf{x}) - \frac{\partial r^{(j)}(\theta,\varphi)}{\partial \varphi}\nabla\varphi(\mathbf{x}). \tag{40}$$

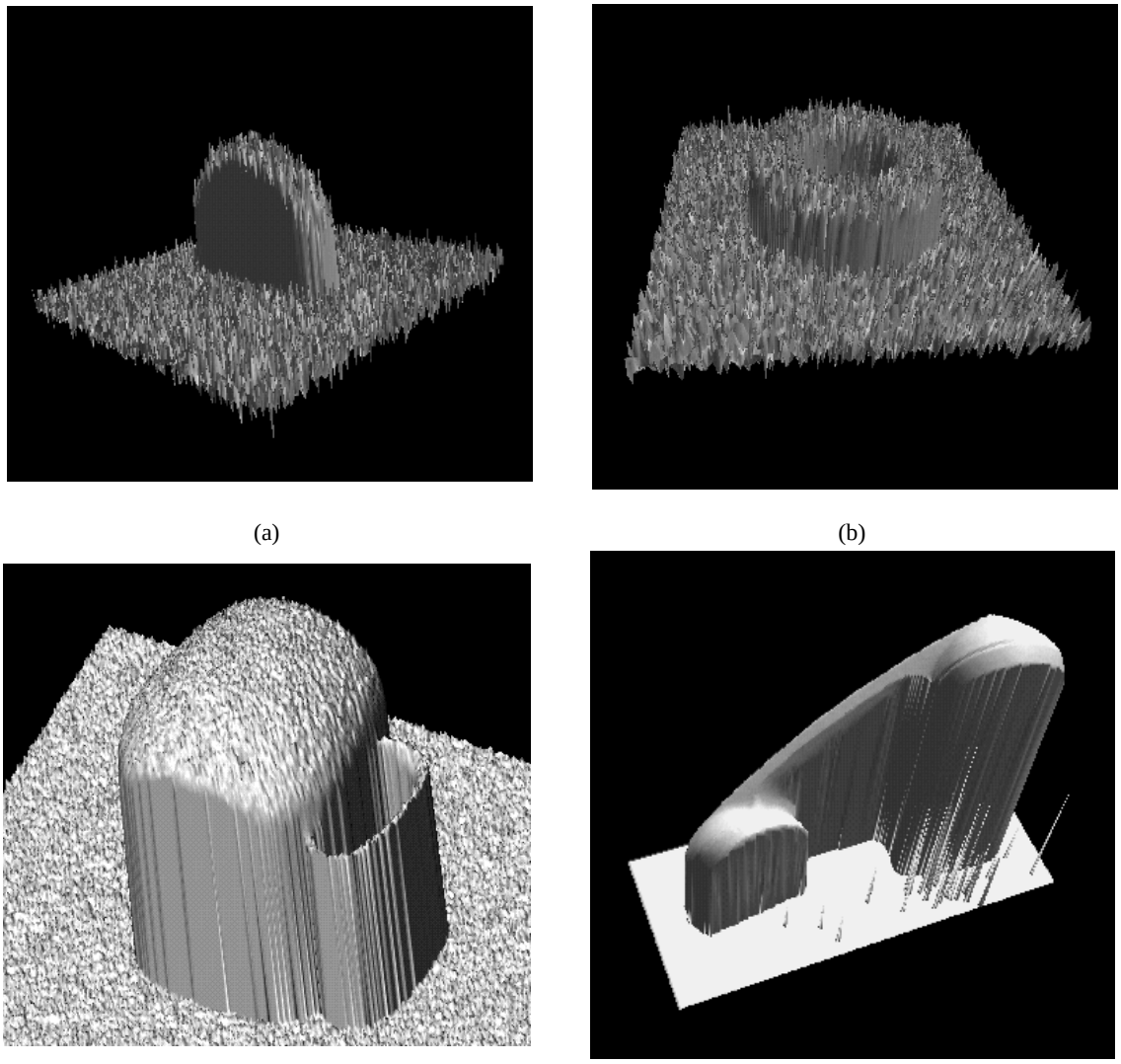
Thus, this formulation has the intuitive result of lowering the confidence near step edges in the range map. Some simple reasoning gives approximate values for $\sigma_{\mathbf{x}}$. If the surface position is known to within one half of a voxel (and voxels have unit length), then one should choose $\sigma_{\mathbf{x}}^2 = 1/12$. If the first-order approximation reduces that by an order of magnitude we have $\sigma_{\mathbf{x}}^2 = 1/120$. Figure 9a shows a noisy range map generated from the CAD model of a mug, and figure 9b shows the resulting confidence map with $\sigma_{\mathbf{x}}^2 = 1/120$ and a constant noise value for all range measurements of 1 unit.

6.1 Results

This section presents 3D reconstructions from several different sets of data. The first is synthetic data of a torus, shown as a surface in Figures 10(a) and 10(b). These 200×200 pixel depth maps are computed analytically and corrupted with 20% uncorrelated Gaussian noise. Six such views of the torus are combined in the examples that follow.

The second set of data is ten synthetic views computed from a CAD model of a mug as shown in figure 10(c). This mug has some smaller features such as the handle (particularly where it attaches to the body) and the top of the rim. Results will be shown with and without 50% additive Gaussian noise.

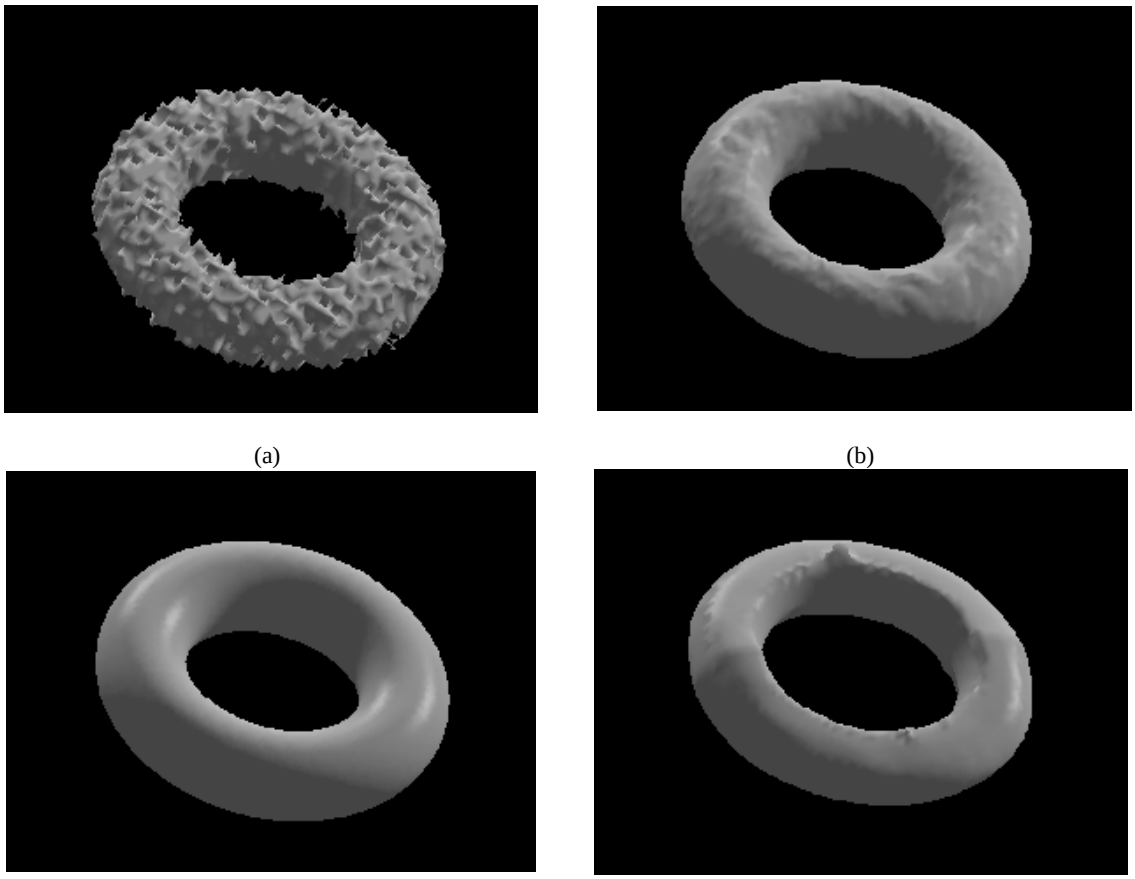
For real data, figure 10(d) shows a 400×500 pixel depth map taken from a telephone receiver using a triangulating laser range finder. Eight such views have been positioned relative to each other, initially by hand and then by an automated surface matching technique (Turk and Levoy 1994).



(a) (b)

(c) (d)

Figure 10: Range maps: Synthetic range data of a torus — 200×200 pixels with 20% Gaussian white noise (as a fraction of smaller diameter) taken of both end (a) and side (b). Synthetic range data of a mug (c) — 256×256 pixels with 50% Gaussian white noise (as a fraction of handle width). Triangulating laser range data of telephone (d).



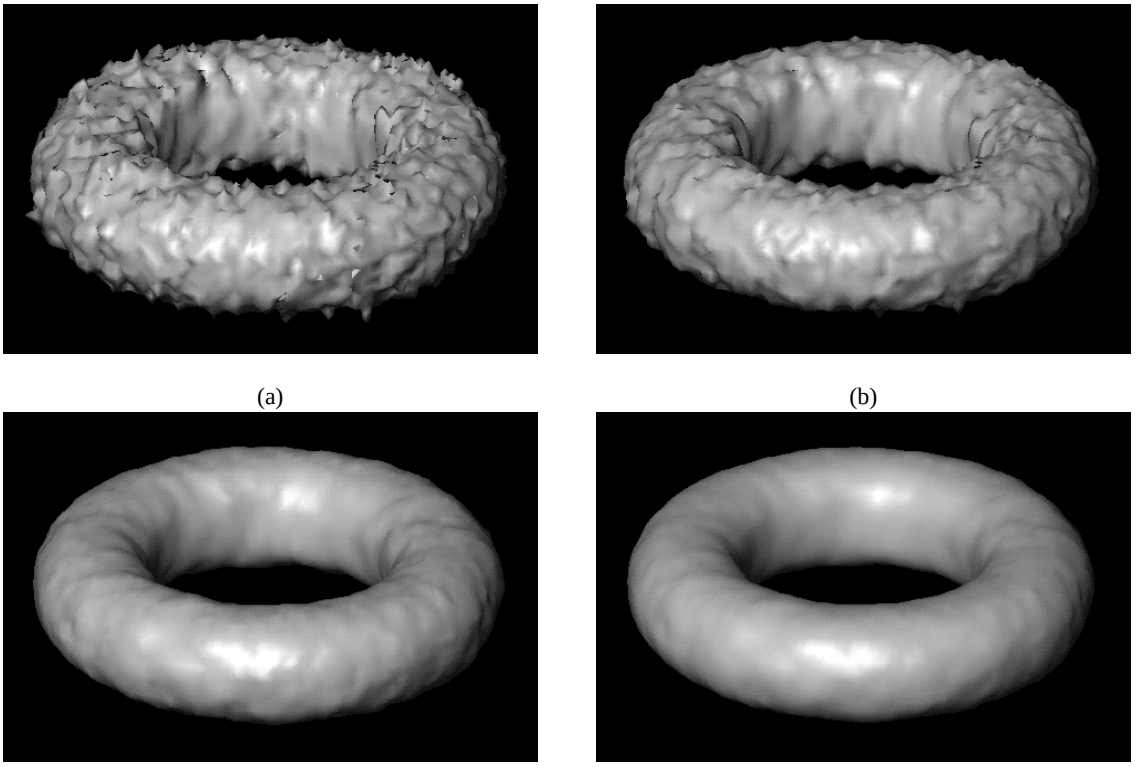
(a) (b)

(c) (d)

Figure 11: An initial model (a) is constructed by combining six points of view of a torus using the zero crossings of $g(\cdot)$. The model which is attracted to the range data but undergoes internal forces evolves and settles into a smoother steady state (b) which resembles the uncorrupted torus (c). Image-based smoothing produces view-dependent artifacts (d).

Figure 11(a) shows the initial model used for fitting the level-set model to the range data for the torus. The initial model is a $80\times80\times40$ voxels and is produced by finding the zero crossings of the data term, $g(\cdot)$. For all of the examples of this paper, 3D surface renderings of level sets are from polygonalized surfaces obtained from the volumes by the marching cubes method (Lorenson and Cline 1982). Figure 11(b) shows the result of the level-set model that uses 11(a) as an initial state and β value of 0.25. The resulting model, which combines the six points of view and the smoothing function, is a reasonable reconstruction of the original object (figure 11(c)). Figure 11(d) shows the result of the combined data term $g(\cdot)$ created by the six noisy torus range maps that have been smoothed with Gaussian blurring prior to their combination. Such an image-space blurring distorts the geometric structure of the range maps so that they no longer fit together; this results in view-dependent artifacts. The object-based smoothing, achieved by incorporating the prior of equation (19) into the level-set approach, does not produce such artifacts.

Figure 12 shows the effects of the parameter β on the final result. The parameter is not a threshold that must be set precisely; results change gradually as β varies over an order of magnitude. Figure 13 shows, for different values of β, the rms error of the reconstruction of the torus as a percentage of the magnitude of the noise added to the synthetic range maps. Even without the prior, the averaging obtained from the MAP method overcomes much of the initial noise in the data. The quality of the reconstruction does depend on the choice of β, but the use of the prior



(a) (b)

(c) (d)

Figure 12: Solutions to reconstruction described in figure 11 with values of β at (a) 0, (b) 0.04, (c) 0.32, and (d) 0.64.

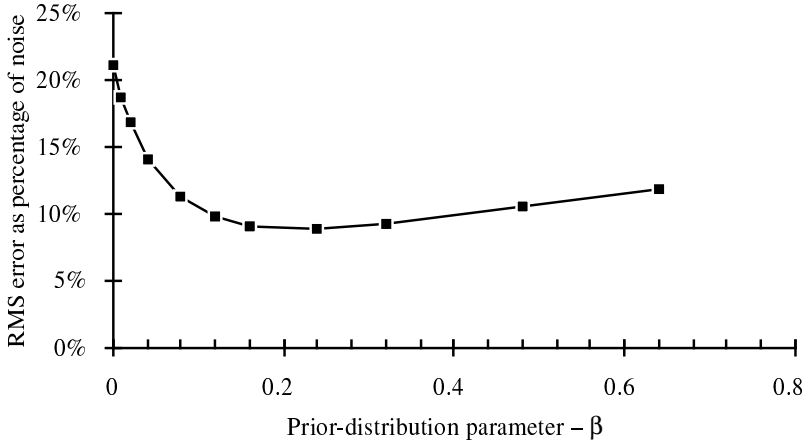


Figure 13: The rms error of the reconstruction of a torus, given as a percentage of the magnitude of the noise, for different values of the prior, β.

provides significant improvements in the reconstruction for any choice of beta that is within an order of magnitude of the optimum.

A 3D model of a mug, shown in figure 14(a), is used to generate 10 range maps. These ten views are then combined to reconstruct the 3D model (figure 14(b)). The reconstruction resembles the original in figure 14(c) to within the accuracy afforded by the discretization of the volume and some small shadowing artifacts around the handle, where it is difficult to completely map because of self occlusions. The final model is $140 \times 140 \times 140$ voxels. At this resolution the handle is less than two voxels thick, but one can still recover the fine details where the handle attaches to the main part of the mug. The facets are in the original, polygonal model. Such large volumes (almost 3 million voxels) necessitate the use of the sparse-field approach, which visits the entire data set only during the initialization, and visits only those voxels that lie near the surface thereafter. With no priors (i.e. $\beta = 0$) and range images that are corrupted by additive Gaussian noise, the surface takes on an rough appearance. Figure 14(d) shows how a small influence of the prior, $\beta = 0.5$, produces a smoother result. The result that includes the smoothing prior is also quantitatively better as determined by the rms error from the original mug model. The $140 \times 140 \times 140$ grid used for the mug required 16 iterations. The entire reconstruction process lasted about 20 minutes on a Sparc 10. Most of that time was spent on the initialization and resampling which requires visiting the entire volume.

For the workstation used in this work, this large model is at the limit of what can be stored in random access memory. As a result, models larger than this typically introduce thrashing and significantly longer computation times. Methods for efficiently representing sparse volumes are well documented in the literature, but access time penalties associated with current technologies (e.g. oct-trees) would make such methods inefficient for the iterative procedures used in this work. Reducing these memory requirements while maintaining run-time efficiency is an area of ongoing investigation.

Figure 15 shows how the algorithm is able to handle outliers. Figure 15(a) shows a single scan from the same mug data set corrupted with 1% replacement noise (i.e. outliers) instead of additive Gaussian noise. The resulting initialization shown in 15(b) is quite poor; it contains hundreds of holes. The combination of smoothing and sub-voxel accuracy obtained from the level-set models pulls the model away from the outliers and gives a convincing reconstruction.

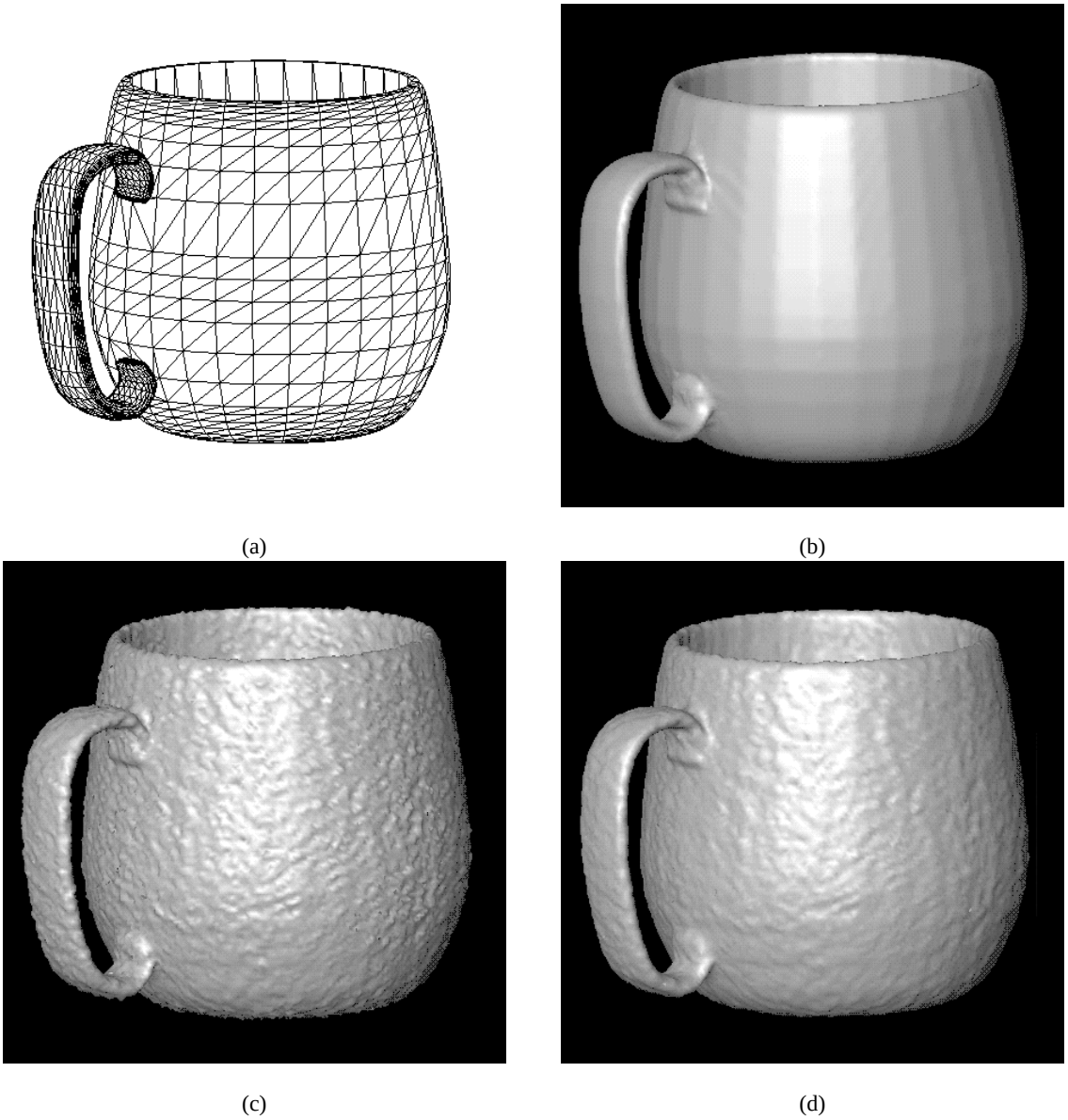
In figure 16 eight range maps of a telephone (taken with a triangulating laser range finder (Curless and Levoy 1995)) are combined to create a 3D reconstruction. Figure 16(a) shows the $40 \times 20 \times 20$ initial model used for fitting level-set models to the range data. That model serves as the initial conditions for the evolution given by equation (39). After the model settles down (change from one iteration to the next drops below a threshold), the volume is resampled onto a finer grid, and the process is repeated. The scanner images have artifacts which affect the result of the modeling as shown in figure 17(a). These artifacts result from false surfaces in the data, as well as misalignments in the range maps. The MAP reconstruction algorithm as it stands cannot realign the data, but to the extent these artifacts have a high-frequency character to them, they can be controlled by adjusting the smoothing parameter, β.

7 Conclusions

This paper has presented a strategy for reconstructing 3D objects by combining range maps taken from different points of view. The strategy is to compute the surface which is mostly likely to have given rise to the data generated by the range scanner; it maximizes the posterior probability of the surface.

By using the independence of the sensor noise and assuming that the collection of data in any one map is dense relative to the object structure, the MAP formulation converts the error from individual range readings into a volume integral. The Euler-Lagrange of that volume integral gives the equation of motion for a surface that seeks to minimize its likelihood conditional on the data. The introduction of a prior leads to a full MAP formulation. This is a fundamental result in 3D reconstruction which poses surface reconstruction as an evolutionary process based on first principles.

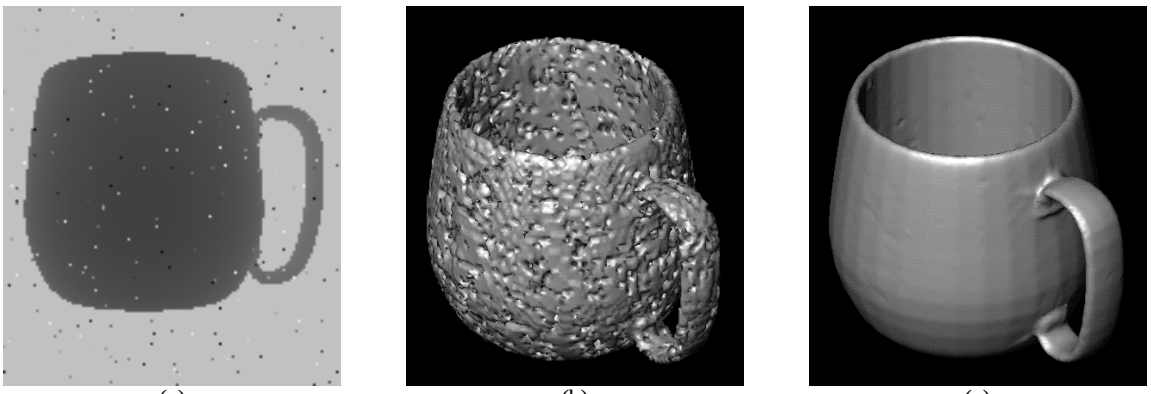
This paper has also proposed level-set models for computing the deformations associated with MAP estimation. The level-set modeling approach is well suited for the MAP reconstruction formulation because it offers flexible



(a) (b)

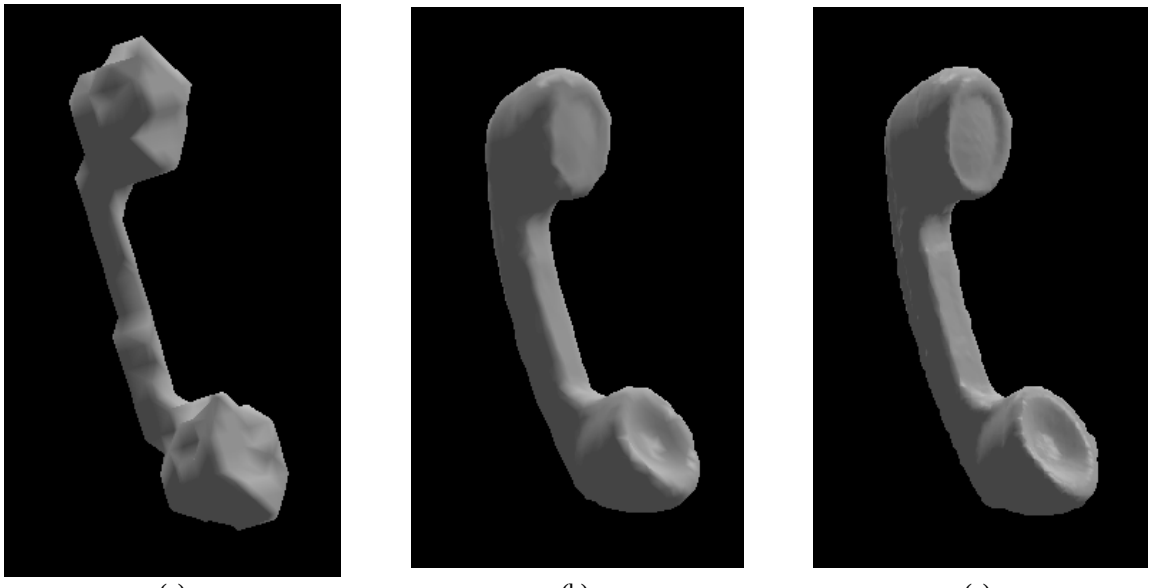
(c) (d)

Figure 14: Level-set MAP reconstruction of a mug using synthetic data generated from a 3D model (a). Without noise the reconstruction (b) is limited only by the resolution of the model ($140 \times 140 \times 140$). With noise, the surface appears rough (c). Including a prior of $\beta = 0.25$ improves the appearance of the reconstruction (d) and the rms error from the original.



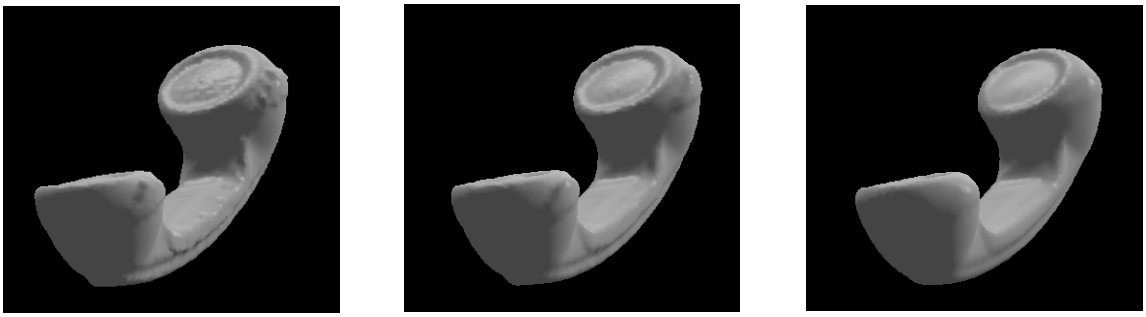
(a) (b) (c)

Figure 15: The effects of outliers: (a) The original mug data set corrupted by replacement noise. (b) The poor initialization that results from that data. (c) The results of the iterative, level-set formulation.



(a) (b) (c)

Figure 16: The results of the level-set MAP reconstruction approach: An initial model (a) is constructed by combining eight points of view of a telephone into a discrete occupancy grid of $40 \times 20 \times 20$ grid points. Results from the course model serve as initial conditions for successively finer models in (b) and (c).



(a) (b) (c)

Figure 17: Artifacts that produce discontinuities can be controlled by tuning the weight β of the smoothing term. In (a), (b), and (c), steady-state solutions with β = 0.1, 1.0, and 5.0, respectively.

topologies, multiscale/multigrid representations, many degrees of freedom, and efficient methods for calculating geometric surface properties. Despite these advantages, the level-set approach as described in the literature suffers from high computational costs and limited accuracy.

This paper has proposed an enhancement to level-set modeling technology in the form of a new algorithm, a sparse-field method. The sparse-field method performs calculations and updates only at those grid point locations that lie near the zero-level set (or any level set of interest). As a result this algorithm is computable in a fraction of the time required for other level-set approaches. This paper has shown that this new method introduces errors that are no worse than previous algorithms. The sparse-field algorithm also enables one to use a first-order approximation to the level-set position, and therefore it is actually more accurate under circumstances where the forcing function is defined to greater accuracy than the grid structure, as is the case with the proposed MAP formulation for 3D reconstruction.

Results on both real and synthetic data confirm the effectiveness of combining the MAP strategy with the level-set modeling method. The use of a very simple prior, which biases solutions toward those that have less surface area, can improve the reconstruction in the presence of uncorrelated noise and high-frequency artifacts.

There are, however, several areas that warrant further development. One area is the problem of large data sets. While the sparse-field method reduces the number of computations required for deforming, it does not alleviate the need to store data for the volume that covers the entire domain. Another area of ongoing investigation is the prior. The second-order smoothing used in this paper tends to distort the shapes of objects when applied too aggressively. This is particularly true when the objects have high curvature, such as the handle of the mug in figure 14. Priors, such as the bending energy, that rely on higher order geometry or priors that incorporate higher level knowledge about the objects could improve the quality of the results.

Acknowledgments

Thanks go to David Elsner for supplying the “mug” data set. The laser range data for the telephone was provided by the Stanford University Computer Graphics Laboratory. This work is supported by the Office of Naval Research grant N00014-97-0227 and the University Research Program in Robotics under the Department of Energy DOE-DE-FG02-86NE37968.

References

Adalstein, D. and Sethian, J. A.: 1995, A Fast Level Set Method for Propogating Interfaces, *Journal of Computational Physics* pp. 269–277.

Alvarez, L., Guichard, F., Lions, P.-L. and Morel, J.-M.: 1992, Axioms and fundamental equations of image processing, *Technical Report 9231*, Ceremade, Universite Paris-Dauphine, Place de Lattre de Tassigny, 75775 Paris Cedex 16 France.

Besl, P. J. and McKay, N. D.: 1992, A method for registration of 3-D shapes, *IEEE Trans. Pattern Anal. Mach. Intelligence* **14**, 239–256.

Bruce, J. and Giblin, P.: 1986, Growth, motion, and 1-parameter families of symmetry sets, *Proc. Roy. Soc. Edinburgh* **104A**, 179–204.

Caselles, V., Kimmel, R. and Sapiro, G.: 1995, Geodesic active contours, *in* ICC (1995), pp. 694–699.

Chen, Y. and Medioni, G.: 1992, Object modeling by registration of multiple range images, *Int. J. Image and Vision Computing* **10**, 145–155.

Chen, Y. and Médioni, G.: 1994, Fitting a surface to 3-d points using an inflating ballon model, *in* A. Kak and K. Ikeuchi (eds), *Second CAD-Based Vision Workshop*, Vol. 13, IEEE, pp. 266–273.

Chien, C.-H. and Aggarwal, J. K.: 1989, Model construction and shape recognition from occluding contours, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11**(4), 372–390.

Curless, B. and Levoy, M.: 1995, Better optical triangulation through spacetime analysis, *in* ICC (1995), pp. 987–994.

Curless, B. and Levoy, M.: 1996, A volumetric method for building complex models from range images, *Computer Graphics (SIGGRAPH '96 Proceedings)* .

DeCarlo, D. and Metaxas, D.: 1995, Adaptive shape evolution using blending, *in* ICC (1995), pp. 834–839.

Dickinson, S., Metaxas, D. and Pentland, A.: 1997, The role of model-based segmentation in the recovery of volumetric parts from range data, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(3), 259–267.

Elfes, A.: 1989, Using Occupancy Grids for Mobile Robot Perception and Navigation, *Computer* **22**(6), 46–57.

Hilton, A., Stoddart, A. J., Illingworth, J. and Windeatt, T.: 1996, Reliable surface reconstruction from multiple range images, *Proceedings from the Sixth European Conference on Compter Vision*, Springer-Verlag.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W.: 1992, Surface reconstruction from unorganized points, *Computer Graphics* **26**(2), 71–78.

ICC: 1995, *Fifth International Conference on Computer Vision*, IEEE Computer Society Press.

Jain, A. and Flynn, P. (eds): 1993, *Three-Dimensional Object Recognition Systems*, Elsevier Science Publishers, The Netherlands.

Jain, R. and Jain, A. (eds): 1990, *Analysis and Interpretation of Range Images*, Springer-Verlag.

Johnson, V. E.: 1993, A framework for incorporating structural prior information into the estimation of medical images, *in* H. H. Barrett and A. F. Gmitro (eds), *Information Processing in Medical Imaging (IPMI'93)*, number 687 in *Lecture Notes in Computer Science*, Springer-Verlag, pp. 307–321.

Kass, M., Witkin, A. and Terzopoulos, D.: 1987, Snakes: Active contour models, *International Journal of Computer Vision* **1**, 321–323.

Kimia, B. J., Tannenbaum, A. R. and Zucker, S. W.: 1992, Shapes, shocks, and deformations I: The components of shape and the reaction-diffusion space, *Technical Report LEMS-105*, Division of Engineering, Brown University.

Lorenson, W. and Cline, H.: 1982, Marching cubes: A high resolution 3d surface construction algorithm, *Computer Graphics* **21**(4), 163–169.

MacDonald, D., Avis, D. and Evans, A.: 1994, Volumetric deformable models: Active blobs, *in* R. A. Robb (ed.), *Visualization In Biomedical Computing 1994*, SPIE—The International Society for Optical Engineering, Mayo Clinic, Rochester, Minnesota, pp. 160–169.

Malladi, R., Sethian, J. A. and Vemuri, B. C.: 1995, Shape modeling with front propogation: A level set approach, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(2), 158–175.

McInerny, T. and Terzopoulos, D.: 1995, Toplogically adaptable snakes, *in* ICC (1995), pp. 840–845.

Miller, J., Breen, D., Lorensen, W., O'Bara, R. and Wozny, M.: 1991, Geometrically deformed models: A method for extracting closed geometric models from volume data, *Computer Graphics (SIGGRAPH '91 Proceedings)* **25**(4), 217–226.

Moravec, H. P.: 1988, Sensor Fusion in Certainty Grids for Mobile Robots, *AI Magazine* **9**(2), 61–77.

Muraki, S.: 1991, Volumetric shape description of range data using "blobby model", *Computer Graphics (SIGGRAPH '91 Proceedings)* **25**(4), 227–235.

Nielson, M.: 1997, Graduated nonconvexity by functional focusing, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(5), 521–525.

Osher, S. and Sethian, J.: 1988, Fronts propogating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations, *Journal of Computational Physics* **79**, 12–49.

Sethian, J. A.: 1996, *Level Set Methods: Evolving Interfaces in Gometry, Fluid Mechanics, Computer Vision, and Material Sciences*, Cambridge University Press.

Snyder, W., Han, Y.-S., Bilbro, G., Whitaker, R. and Pizer, S.: 1995, Image relaxation: restoration and feature extraction, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(6), 620–624.

Staib, L. and Duncan, J.: 1992, Boundary finding with parametrically deformable models, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**, 1061–1075.

Szeliski, R., Tonnesen, D. and Terzopoulos, D.: 1993, Modeling surfaces of arbitrary topology with dynamic particles, *Proceedings Fourth International Conference on Computer Vision (ICCV'93)*, IEEE Computer Society Press, Berlin, Germany, pp. 82–87.

Terzopoulos, D., Witkin, A. and Kass, M.: 1988, Constraints on deformable models: Recovering 3d shape and nonrigid motion, *Artificial Intellegence* **36**(1), 91–123.

Turk, G. and Levoy, M.: 1994, Zippered polygon meshes from range images, *SIGGRAPH '94 Proceedings*, pp. 311–318.

Whitaker, R. T.: 1994, Volumetric deformable models: Active blobs, *in* R. A. Robb (ed.), *Visualization In Biomedical Computing 1994*, SPIE, Mayo Clinic, Rochester, Minnesota, pp. 122–134.

Whitaker, R. T.: 1995, Algorithms for implicit deformable models, *in* ICC (1995).

Whitaker, R. T. and Chen, D. T.: 1994, Embedded active surfaces for volume visualization, *SPIE Medical Imaging 1994*, Newport Beach, California.

Zhang, Z.: 1994, Iterative point matching for registration of free-form curves and surfaces, *Int. J. Computer Vision* **13**, 119–152.

Appendix

A formal definition of active sets and the operations performed on them gives some general properties regarding their behavior during the sparse-field algorithm. Let u be a discrete approximation to ϕ on a rectangular grid. On each of the grid points (or voxels in 3D), u has a value equal to that of ϕ. An active set is a set of grid points that are adjacent to the level set. For notational convenience the zero-level set is used, and thus active sets consist of grid points that lie near zero crossings. As the level set moves, two things happen to active sets. First, the values of active set and nearby points change. Second, points are added and removed from the active set, i.e., activity is *passed* on from one grid point to another. The remainder of this section serves to formalize these ideas and show that active sets maintain their important properties as the sparse-field algorithm progresses. In order to maintain generality, the discussion proceeds in n dimensions, where $n = 3$ for surface reconstruction problem at hand.

A volume $I = \{X, u\}$ of dimension n is a set of grid points, X, arranged in a rectilinear grid and a discrete mapping $u : X \mapsto \mathbb{R}$. A grid point $x_j \in X$ has an associated index $i(x_j) \in \mathrm{I}^n$ (i.e., $i = i_1, \ldots, i_n$ and $i_j \in \mathrm{I}$). This index has a basis, $e_1, \ldots, e_n$, which consists of unit vectors along the grid lines, i.e., $e_1 = (1, 0, \cdots), e_2 = (0, 1, 0, \cdots), \cdots, e_n = (0, \cdots, 1)$.

Each grid point has a set of $2n$ connected neighbors which are given by $C(x_j) = \{x_k | i(x_k) = i(x_j) \pm e_m, 1 <= m <= n\}$. That is, the $2d$ neighbors of x_j, given by $C(x_j)$, are those points that lie adjacent to x_j in one of the grid directions.

One minor point is the handling of boundary conditions on the domain. For this work we assume that every grid point has $2d$ adjacent grid points, meaning either a toroidal topology (wrap around) or an infinite array of grid points. All of the results described in this section can be extended to other topologies or boundary conditions.

The *adjacent* operator, $A(x_i, x_j)$ for $x_i, x_j \in X$, indicates whether or not two or more grid points are adjacent, i.e.,

$$A(x_i, x_j) \stackrel{\triangle}{=} x_j \in C(x_i). \tag{41}$$

The sign operator $\mathrm{O}\,(x_i, x_j, u)$ indicates whether or not the values of u associated with two grid points have values of opposite sign, i.e.,

$$\mathrm{O}\,(x_i, x_j, u) \stackrel{\triangle}{=} \left(u_{x_i} \geq 0 \text{ and } u_{x_j} 0\right) \text{ or } \left(u_{x_i} < 0 \text{ and } u_{x_j} \geq 0\right). \tag{42}$$

Note that both $\mathrm{O}\,(x_i, x_j)$ and $A(x_i, x_j, u)$ are commutative. An adjacent grid point pair with values that have opposite signs are said to lie on a *zero crossing*.

An *active set* $\mathcal{A}$ is a subset of X such that for every adjacent pair with opposite sign, at least one of the grid points of that pair is in the active set. That is, $\mathcal{A} \subset X$ is an active set if and only if $\forall x_i, x_j \in X$, $O(x_i, x_j, u)$ and $A(x_i, x_j) \implies \{x_i, x_j\} \cap \mathcal{A} \neq \emptyset$. This definition means that an active set could include grid points that are not necessary to satisfy the condition. An active set is said to be *efficient* if for all members of the active set have a neighbor of opposite sign, i.e., $\mathcal{A}$ is efficient if and only if $\forall x_j \in \mathcal{A}\ \exists\ x_i$ s.t. $O(x_i, x_j, u)$ and $A(x_i, x_j)$. Efficient active sets consist entirely of grid points that lie on zero crossings.

There are several properties of active sets that are important. First, an efficient active set can be constructed from a volume (if it is finite) by testing the values of each grid point and its neighbors and constructing a union of all of those grid points that lie on zero crossings. Second, an efficient active set can be constructed from any (finite) active set by successively removing those grid points that do not lie on zero crossings. Third, active sets divide images into enclosed positive and negative regions. This is true because, by the definition of an active set, there is no connected path (a sequence of adjacent grid points) from a grid point with positive value to a grid point with negative value that does not pass through an active grid point.

The sparse field algorithm works with active sets and performs two distinct actions on these sets; it moves the activity of grid points to adjacent grid points, and it changes the values of grid points. The following results show that sparse-field algorithm maintains the properties of active sets (they continue to divide positive and negative regions).

A *movement* of an active set $\mathcal{A}$ is a procedure for constructing a new set of grid points $\mathcal{A}' = M(\mathcal{A}, I)$. This procedure is to remove a grid point $x_j \in \mathcal{A}$ from $\mathcal{A}$ and add to $\mathcal{A}$ all of x_j's neighbors that have opposite sign. That

is,

$$M(\mathcal{A}, I) \stackrel{\triangle}{=} \{\mathcal{A} - \{x_j\}\} \cup \{x_i | A(x_i, x_j) \text{ and } O(x_i, x_j, u)\}. \quad (43)$$

Proposition 1 *A set of grid points $\mathcal{A}'$ constructed by a movement procedure, $\mathcal{A}' = M(\mathcal{A}, I)$, is an active set.*

Proof Proposition 1 Let x_j denote the grid point that is removed. Assume that $\mathcal{A}'$ is not an active set. Then there exists a pair of grid points $\{x_i, x_k\}$ such that $O(x_i, x_j, u)$ and $A(x_i, x_k)$ and $\{x_i, x_k\} \cap \mathcal{A}' = \emptyset$. There are two cases to consider:

- $x_j \notin \{x_i, x_k\}$ In this case neither of the grid points $\{x_i, x_k\} \cap \mathcal{A} = \emptyset$. Thus $\mathcal{A}$ is not an active set, which violates one of the assumptions.
- $x_j \in \{x_i, x_k\}$ This also violates the assumption because all opposite-sign neighbors of x_j are in $\mathcal{A}'$.

□

Proposition 2 *If $\mathcal{A}$ is an efficient active set, then the set of grid points $\mathcal{A}'$ constructed by a movement procedure $\mathcal{A}' = M(\mathcal{A}, I)$, is also an efficient active set.*

Proof Proposition 2 Let x_j denote the grid point that is removed. $\mathcal{A}'$ is an active set by Proposition 1. Thus only the proof of efficiency is necessary. Assume that $\mathcal{A}'$ is not efficient, then there exists an active grid point x_i with a neighborhood $N = \{x_k | A(x_i, x_k\}$ such that $\{x_k | O(x_i, x_k, u)\} \cap N = \emptyset$. There are two cases to consider:

- $x_j \in \mathcal{A}$: This violates the assumption because $\mathcal{A}$ is assumed to be efficient.
- $x_j \notin \mathcal{A}'$: In this case x_j is one of the grid points that is added to $\mathcal{A}'$ by the movement procedure. However, these grid points are added because they have a neighbor of opposite sign, namely x_j.

□

An *update* of an active set is the construction of a new image which has a new value for one of its active points, i.e., $I' = \{X, u'\}$ where $u' = U(u, \mathcal{A})$.

Proposition 3 *An update of I to I' with $I' = \{X, u'\}$ and $u' = U(u, \mathcal{A})$ preserves the active set $\mathcal{A}$, i.e., $\mathcal{A}$ is still an active set of I'.*

Proof Proposition 3 Let x_j be the grid point with a value that is changed. Assume that $\mathcal{A}$ is not an active set of I'. Then there exists a pair of grid points $\{x_i, x_k\}$ such that $O(x_i, x_j, u')$ and $A(x_i, x_k)$ and $\{x_i, x_k\} \cap \mathcal{A} = \emptyset$. There are two cases to consider:

- $x_j \notin \{x_i, x_k\}$ In this case neither of the grid points $\{x_i, x_k\} \cap \mathcal{A} = \emptyset$, which means that $\mathcal{A}$ is not an active set of X, u (since u and u' differ only in the value at x_j), which violates one of the assumptions.
- $x_j \in \{x_i, x_k\}$ This also violates the assumption because x_j is by definition in the active set $\mathcal{A}$.

□

It is provable that the update procedure does not preserve the efficiency of an active set in all cases. Certain combinations of grid points can be shown to produce inefficient active sets after an update procedure. Pathological cases can produce dense sets of active points that fill whole regions and render the sparse-field method inefficient.

One solution to the problem of inefficiency is a *pruning* process which systematically removes unnecessary active points. A pruning procedure $P(u, \mathcal{A})$ is a procedure that produces a new active set $\mathcal{A}' = \mathcal{A} - \{x_j\}$ if the neighborhood N of x_j is of the same sign, i.e., $O(x_i, x_j, u) \forall x_i \in N$, and $\mathcal{A}' = \mathcal{A}$ otherwise. The pruning procedure can be repeated for each $x_j \in \mathcal{A}$ after an update step and can be shown to produce an efficient active set. The author has found that in practice such a pruning procedure is unnecessary, and that under normal situations active sets remain efficient except at singularities, i.e., where level sets break, join, or disappear.

Reducing Aliasing Artifacts in Iso-Surfaces of Binary Volumes

Ross T. Whitaker [1]

School of Computing, University of Utah

Appears in: *IEEE Volume Visualization and Graphics Symposium*, October 2000, pp. 23–32.

Abstract

This paper presents a method for filtering binary volumes in order to reduce the aliasing artifacts that typically result when trying to visualize the surfaces described by such a binary partitioning. The proposed method is a reformulation of the technique of *constrained elastic surface nets* from the literature. The strategy is to estimate a 3D surface using an iterative relaxation process, which finds the surface of minimal area while treating the binary input data as a set of constraints. Unlike surface nets, the proposed method does not require the generation of any explicit surface representation; it acts directly on the volume. This is done by treating the level sets of the volume as deformable surfaces moving under a mean curvature flow constrained by a discretely sampled inside/outside function. Because the proposed method does not require generating a surface mesh, it avoids the various singularities and topological problems associated with iso-surface generation. It also avoids the artifacts that an irregular mesh topology can produce in deformation processes. If the goal is volume rendering, the proposed method avoids the time consuming process of converting surface meshes back into volumes.

Keywords: Antialiasing, Binary Volumes, Level Sets.

1 Introduction

The greatest drawback of using volumes to implicitly model 3D objects is their limited resolution. For instance, a volume is $f_{i,j,k}$ or $f : \mathcal{D} \mapsto \mathbb{R}$ where $\mathcal{D}$ is the discrete domain, i.e., $\mathcal{D} = \mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$ and $\mathcal{X} = \{1, 2, \ldots, N_x\}$, $\mathcal{Y} = \{1, 2, \ldots, N_y\}$, and $\mathcal{Z} = \{1, 2, \ldots, N_z\}$. The volume f is regarded as a discrete sampling of a continuous function $\phi : D \mapsto \mathbb{R}$, where the domain $D \subset \mathbb{R}^3$ corresponds to the extent of the 3D space in which the surfaces are modeled.

[1] School of Computing, Merrill Engineering Building, 50 Central Campus Drive, Salt Lake City, UT 84112-9205; Email: whitaker@cs.utah.edu.

An implicit surface model is an iso-surface of ϕ, which is only approximated by $f_{i,j,k}$. The particular iso-value that is being considered is a matter of convention, and for notational convenience I consider only the zero crossings of ϕ for discussions that follow. Thus, the surface is the set of points

$$\mathcal{S} = \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \middle| \, \phi(x, y, z) = 0 \right\}. \tag{1}$$

The zero-set of $\phi(x, y, z)$ is not constrained to pass through the grid points, $\mathcal{D}$; it could wind its way between grid points in any kind of arbitrary fashion.

The process of estimating the iso-surfaces of ϕ from the discrete sampling $f_{i,j,k}$ is *generally* ill posed— a discrete sampling does not give enough information to recover ϕ or its iso-surfaces. One typical strategy for finding the iso-surfaces of ϕ is to interpolate $f_{i,j,k}$. The accuracy of that interpolation depends on two things: the grid spacing and the quality with which the interpolant fits the underlying data, ϕ.

When using volumes as a modeling paradigm to describe a particular surface shape, the choice of ϕ, the embedding, is not unique—there are infinitely-many choices. A sensible strategy is to choose an embedding which has the least artifacts when sampled on a discrete grid. For a given surface $\mathcal{S}$ that is being modeled by an implicit function, ϕ, sampled with a finite resolution, some choices of ϕ are inherently better than others. The best embeddings are those that are flat in the direction perpendicular to the surface, i.e., the second- and higher-order derivatives in that direction are zero. Indeed, several authors [**?**, **?**] have noted that the distance transform yields good results for both surface extraction and volume rendering. The worst example of an embedding is a binary volume, that is, a volume that contains two values: one value indicating voxels that are outside the object and the other indicating voxels that are inside.

Binary volumes, however, are interesting for several practical reasons. First, in some cases, such as medical imaging, a volume dataset can be segmented to produce a set of voxels that correspond to some particular object (or anatomy). This segmentation can be manual, in which case a user would identify (usually with the aid of a GUI) all of the voxels that belong to a certain object. The segmentation could also be automated, relying on methods such as pattern classification, flood fill, and morphological watersheds, which produce segmentations that are *hard*, i.e., binary. Binary volumes are also important when using a 3D imaging device that produces data very high contrast. In such cases the measured data is *essentially* binary with regard to both the information it contains and the problems it presents in rendering. Binary volumes can be important when visualizing mathematical expressions, such as fractals, that

cannot be evaluated as continuous functions. Binary volumes are also interesting because they require so little data, and, with the use of run length encoding, are very well suited to compression.

This paper proposes a method for extracting iso-surfaces from binary volumes, which significantly reduces aliasing artifacts while retaining important features of objects in the volume. The proposed method follows from first principles; it incorporates *no* free parameters and guarantees a level of fidelity to the original data.

2 Related Work

Most of the literature that addresses aliasing artifacts in volumes falls into one of two strategies: filtering and surface fitting. The prevailing strategy is to filter the data, typically with some stationary, low-pass, isotropic kernel and then perform a conventional volume rendering (or surface extraction) on the *smoothed* data [**?**, **?**]. There are two basic problems with this approach. The kernel used to smooth the volumes must be large enough to "bridge" the discrete steps that binary volumes introduce into otherwise smooth surfaces. Such large kernels tend to blur away details that are unrelated to aliasing that might be important aspects of the structures being modeled. Also, the low-pass filtering approach leaves open a number of arbitrary decisions, regarding the shape and size of the smoothing filter, which can significantly affect the final result. One of the advantages of the proposed method is that it introduces virtually no free parameters—it is a closed algorithm that does not require tuning.

Another closely related body of research is the work on *optimal reconstruction* of continuous functions from discrete samples (sometimes called *linear systems*). A nice review with some novel schemes for volumes is given by Machiraju and Yagel [**?**]. Given an infinite set of discrete samples of a band-limited signal, the continuous function can be *completely recovered* from the sampling by convolving with the sinc function. Of course, "undoing" the sampling process could, in principle, eliminate aliasing artifacts (for grey-scale volumes). There are several issues with this strategy. First, volumes are rarely infinite, and therefore the sinc function must be truncated. Thus the results are not ideal, even if the band-limited assumption holds, and one typically combines this with low-pass filtering, which further reduces fidelity. The reconstruction approach is particularly handicapped for binary volumes, because they so clearly violate the band-limited assumption.

The second basic strategy for reducing aliasing, particularly in binary volumes is surface fitting. In the context of ray casting, surface fitting can be achieved by a root trapping approach along the viewing ray with appropriate interpolation of grey-scale values and other volume attributes, as shown by [**?**]. This reduces stair-stepping artifacts, but only within the kernel over which the interpolant is defined, and practical considerations limit these kernels to 2–5 voxels. Alternatively, one can define a parametric surface model that moves toward the zero crossings (or any appropriate iso-value) of the original data. This movement usually takes the form of an incremental minimization that combines the distance to the surface with some internal constraints [**?**, **?**].

The work presented in this paper is most closely related to the work of Gibson [**?**], who uses a deformable-surface approach to reducing aliasing artifacts. Gibson sheds new light on the problem of extracting surfaces from binary volumes with an insightful strategy: *treat the binary data as a constraint on a surface that is subject to a regularization process*. Gibson proposes a several step algorithm called *constrained surface nets*, which embodies this strategy. The algorithm begins by extracting a surface mesh from the volume. This initial mesh consists of a vertex in each cell (an eight-voxel neighborhood arranged in a cube) whose corners indicate a transition from inside to out. This mesh undergoes an iterative process of deformation, where each vertex moves to the mean of its neighbors but is prohibited from moving outside of its original cell. The resulting surface can be converted back into a volume by computing a discrete sampling of the distance transform to the set of triangles associated with the final mesh.

Constrained surface nets have some very useful properties. They are essentially the solution of a constrained minimization of surface area. Constrained surface nets are capable of creating flat surfaces through sequences of distant "terraces" or "jaggies", which are not easily spanned by a filter or interpolating function. The final solution is guaranteed to lie within a fixed distance of the original mesh, thus preserving small details, and even discontinuities, that can be lost through other anti-aliasing techniques.

The proposed method borrows from the basic philosophy of constrained surface nets, but it eliminates the need for an intermediate surface mesh—it operates directly on the volume. It is a transformation of a binary volume to a grey-scale volume, and thus it is a kind of nonlinear filtering process. The zero-set of the volume that results from the algorithm has the desirable properties of the constrained surface net. However, the algorithm makes no explicit assumptions about the topology of the surface, but instead allows the topology of the surface to develop from the constrained minimization process.

3 Level-Set Models

This paper uses a deformable surface model which is represented implicitly, as a level set of a grey-scale volume. Osher and Sethian [**?**] introduced level sets as a mechanism for computing moving wave fronts, and they describe numerical schemes for computing the differential equations that result from this formulation. This paper gives only a brief introduction to level-set methods; a more comprehensive review of level-set methods is given by [**?**].

The strategy of level-set models is to represent surface movements implicitly, via changes in a discrete approximation to a time-varying embedding, $\phi(\bar{x}, t)$. Consider a point on a, $\bar{S}(t)$ on a surface, $\mathcal{S}$, defined implicitly as the k level-set of $\phi(\bar{x}, t)$. The surface and the embedding both change over time such that $\mathcal{S}$ remains the k level set of $\phi(\bar{x}, t)$. The precise relationship between the surface movements and changes in the embedding is given by the chain rule. Because k is constant the total derivative with respect to t is zero,

$$\frac{\mathrm{d}\phi(\bar{S}(t), t)}{\mathrm{d}t} = \nabla\phi(\bar{S}(t), t) \cdot \frac{\mathrm{d}\bar{S}(t)}{\mathrm{d}t} + \frac{\partial\phi(\bar{S}(t), t)}{\partial t} = 0, \quad (2)$$

where $\nabla\phi$ is the vector of partial derivatives in space,

$$\nabla\phi \equiv (\partial\phi/\partial x,\ \partial\phi/\partial y,\ \partial\phi/\partial z). \quad (3)$$

Thus changes in the embedding are given by

$$\frac{\partial\phi(\bar{S}, t)}{\partial t} = -\nabla\phi(\bar{S}, t) \cdot \frac{\mathrm{d}\bar{S}}{\mathrm{d}t}. \quad (4)$$

To complete the formulation, express the surface movements in terms that depend only on the surface position and its local geometric structure. Such quantities are independent of the representation of $\mathcal{S}$ and can, in turn, be expressed in terms of the embedding, ϕ. This removes $\bar{S}$ from equation **??**, leaving a partial differential equation on ϕ. In previous work [**?**], the author gives a table of different terms for surface deformations and their level-set equivalents.

4 Estimating Surfaces From Binary Data

Binary volumes are often visualized by treating them as implicit functions and rendering the surfaces that correspond to the zero-sets of interpolated version of $B : \mathcal{D} \mapsto \{-1, 1\}$, the binary volume. Alternatively, one could treat a binary volume, regardless of

its origins, as a threshold (or binarization) of a discrete sampling of an embedding. That is,

$$\phi(x,y,z) \overset{\text{descretization}}{\longrightarrow} f_{i,j,k} \overset{\text{threshold}}{\longrightarrow} B_{i,j,k} \tag{5}$$

From this point of view, the problem of *extracting* surfaces from a binary volume is really the problem of *estimating* either f or ϕ, and extracting surfaces from one of those functions. However, the loss of information (i.e. projection) associated with the binary sampling leaves the inverse problem ill posed—that is, for a given binary volume there are infinitely many embeddings from which it could have been derived.

The strategy in this paper is to construct a discrete sampling of a ϕ that *could* have given rise to B. An estimate of the embedding, $\hat{\phi}$, is *feasible* if

$$\hat{\phi}(\bar{x})B_{\bar{x}} \geq 0 \; \forall \; \bar{x} \in \mathcal{D}, \tag{6}$$

i.e., $B_{\bar{x}}$ and $\hat{\phi}(\bar{x})$ must have the same sign at the grid points of $\mathcal{D}$. This is the same as saying that the zero-set of $\hat{\phi}$ must enclose all of those points indicated by the binary volume as *inside* and none of the points that are *outside*.

The ill-posed nature of the problem is addressed by imposing some criterion, a regularization, to which $\hat{\phi}$ must conform. In the case of surface estimation, a natural criterion is to choose the surface with minimal area. Often, but not always, surfaces with less area are qualitatively *smoother*. In the level-set formulation, the combined surface area of all of the level sets of ϕ is the integral of the level-set density (which is the gradient magnitude of ϕ) over the domain D. Thus, using level sets, the constrained minimization problem for reconstructing surfaces from binary data is

$$\begin{aligned} \hat{\phi} \quad = \quad & \arg\min_{\phi} \left[\int_D |\nabla \phi(\bar{x})| \, \mathrm{d}\bar{x} \right], \\ & \text{such that } \; \phi(\bar{x})B_{\bar{x}} \geq 0 \; \forall \; \bar{x} \in \mathcal{D}. \end{aligned} \tag{7}$$

Using the method of undetermined multipliers, construct the Lagrangian:

$$F(\phi, \bar{\lambda}) = \int_D |\nabla \phi(\bar{x})| \, \mathrm{d}\bar{x} + \sum_{\bar{x}_i \in \mathcal{D}} \lambda_i \phi(\bar{x}_i) B_{\bar{x}_i}, \tag{8}$$

where $\lambda_i \leq 0$.

The Kuhn-Tucker [**?**] conditions describe the behavior of the solution. Using the first variation and letting $\bar{\lambda} = \lambda_1, \lambda_2, \ldots$, this gives

$$\frac{\partial F\left(\phi(\bar{x}), \bar{\lambda}\right)}{\partial \lambda_i} = \phi(\bar{x}_i)B(\bar{x}_i) = 0 \; \forall \; \bar{x}_i \in \mathcal{D}, \; \lambda_i < 0 \tag{9}$$

$$\frac{\partial F\left(\phi(\bar{x}), \bar{\lambda}\right)}{\partial \lambda_i} = \phi(\bar{x}_i)B(\bar{x}_i) \geq 0 \; \forall \; \bar{x}_i \in \mathcal{D}, \; \lambda_i = 0 \tag{10}$$

$$\frac{\partial F\left(\phi(\bar{x}), \bar{\lambda}\right)}{\partial \phi(\bar{x})} = H(\bar{x}) = 0 \; \forall \; \bar{x} \notin \mathcal{D} \tag{11}$$

$$\frac{\partial F\left(\phi(\bar{x}), \bar{\lambda}\right)}{\partial \phi(\bar{x})} = H(\bar{x}_i) + \lambda_i B(\bar{x}_i) = 0 \; \forall \; \bar{x}_i \in \mathcal{D} \tag{12}$$

where H is the gradient magnitude times the mean curvature of ϕ , which is documented in the literature [**?**, **?**] to be

$$H = \frac{\begin{array}{c}(\phi_y^2 + \phi_z^2)\phi_{xx} + (\phi_x^2 + \phi_z^2)\phi_{yy} \\ +(\phi_x^2 + \phi_y^2)\phi_{zz} - 2\left(\phi_x\phi_y\phi_{xy}\right. \\ \left.+\phi_x\phi_z\phi_{xz} + \phi_y\phi_z\phi_{yz}\right)\end{array}}{\phi_x^2 + \phi_y^2 + \phi_z^2}. \tag{13}$$

The subscripts with respect to variables represent partial derivatives with respect to those variables, for example, $\phi_x = \partial\phi(x,y,z)/\partial x$.

These conditions describe the properties of the solution. For points in the domain that are not on the grid, the level sets of the solution are flat or hyperbolic (saddle points) with the principle curvatures offsetting one another. For points in the domain that fall on one of the grid points, there are two cases: the level sets of ϕ are convex at places with $B(\bar{x}_i) > 0$ and concave where $B(\bar{x}_i) < 0$, which is consistent with a solution that is "stretched" around the positive and negative constraints. Also when the curvature is non-zero at a grid point,

$$B(\bar{x}_i)\phi(\bar{x}_i) = \pm\phi(\bar{x}_i) = 0, \tag{14}$$

which means that the zero-set falls through grid points of $\mathcal{D}$ except in those areas where the solution is flat.

This analysis leads to a gradient-descent strategy, with an evolution parameter t. The values of the λ_i's, depend on the curvature of the solution, and cannot be known a priori. However their signs are known, and their magnitudes must be sufficient to offset the curvature of the level-set at $\phi(\bar{x}, t) = 0$. Therefore starting with an initial estimate that is feasible, one can update ϕ in such a way that it minimizes the surface area but does not violate the constraints:

$$\frac{\partial \phi}{\partial t} = \begin{cases} 0 & \text{For } \bar{x} = \bar{x}_i \in \mathcal{D}, \; \phi(\bar{x}) = 0, \\ & \text{and } \; H(\bar{x})B_{\bar{x}} > 0 \\ H(\bar{x}) & \text{otherwise} \end{cases}. \tag{15}$$

5 Implementation

The solutions to the partial differential equations described in the previous section are computed using finite differences on a discrete grid—typically, but not necessarily, the same grid, $\mathcal{D}$, on which the binary function $B_{i,j,k}$ is sampled. Derivatives in equations **??** and **??** are computed using centralized differences, as documented in the literature [**?**]. Let $u^n_{i,j,k}$ be a discrete approximation to $\phi(\bar{x}, t)$ at the nth discrete time step. This gives

$$u^{n+1}_{i,j,k} = u^n_{i,j,k} + \Delta t \Delta u^n_{i,j,k}, \tag{16}$$

where $u^0_{i,j,k}$ is the initial estimate, Δt is a constant that is chosen to ensure stability, and $\Delta u^n_{i,j,k}$ is the discrete approximation to $\partial\phi/\partial t$. Assume, without a loss in generality, that the grid spacing is unity. For centralized differences the derivative operators are defined in a way that provides the "tightest" derivative, i.e.,

$$\begin{aligned} \phi_x &\approx \delta_x u^n_{i,j,k} = \frac{u^n_{i+1,j,k} - u^n_{i-1,j,k}}{2}, \\ \phi_y &\approx \delta_y u^n_{i,j,k} = \frac{u^n_{i,j+1,k} - u^n_{i,j-1,k}}{2}, \\ &\vdots \\ \phi_{xx} &\approx \delta_{xx} u^n_{i,j,k} = u^n_{i+1,j,k} + u^n_{i-1,j,k} - 2u^n_{i,j,k}, \\ &\vdots \\ \phi_{xz} &\approx \delta_{xz} u^n_{i,j,k} = \delta_x \delta_z u^n_{i,j,k}, \end{aligned} \tag{17}$$

and so forth.

These derivatives lead directly to the calculation of the discrete approximation of the mean curvature, from equation **??**. The following update maintains the constraint at each iteration:

$$u^{n+1}_{i,j,k} = \begin{cases} \max\left(u^n_{i,j,k} + \Delta t H^n_{i,j,k}, 0\right) & B_{i,j,k} = 1 \\ \min\left(u^n_{i,j,k} + \Delta t H^n_{i,j,k}, 0\right) & B_{i,j,k} = -1 \end{cases}, \tag{18}$$

where

$$H^n_{i,j,k} = \frac{\begin{array}{c}\left(\left(\delta_y u^n_{i,j,k}\right)^2 + \left(\delta_z u^n_{i,j,k}\right)^2\right)\delta_{xx} u^n_{i,j,k} \\ + \left(\left(\delta_x u^n_{i,j,k}\right)^2 + \left(\delta_z u^n_{i,j,k}\right)^2\right)\delta_{yy} u^n_{i,j,k} \\ + \left(\left(\delta_x u^n_{i,j,k}\right)^2 + \left(\delta_y u^n_{i,j,k}\right)^2\right)\delta_{zz} u^n_{i,j,k} \\ -2\left(\delta_x u^n_{i,j,k}\,\delta_y u^n_{i,j,k}\,\delta_{xy} u^n_{i,j,k}\right. \\ +\delta_x u^n_{i,j,k}\,\delta_z u^n_{i,j,k}\,\delta_{xz} u^n_{i,j,k} \\ \left. +\,\delta_y u^n_{i,j,k}\,\delta_z u^n_{i,j,k}\,\delta_{yz} u^n_{i,j,k}\right)\end{array}}{\left(\delta_x u^n_{i,j,k}\right)^2 + \left(\delta_y u^n_{i,j,k}\right)^2 + \left(\delta_z u^n_{i,j,k}\right)^2}, \qquad (19)$$

To ensure stable solutions, the constant, Δt, is determined from the literature [**?**, **?**] to be $1/6$.

Thus the strategy for estimating surfaces from binary volumes is as follows. Start with an estimate for ϕ that satisfies the constraint, such as the original binary volume. Calculate a sequence of $u^n_{i,j,k}$'s according to equations **??** and **??**, defining the "final" solution to be the $u^n_{i,j,k}$ whose values represent an insignificant change from the previous solution.

While the algorithm described above is sound, it carries with it a significant computational burden; it is an iterative scheme that requires a sequence of updates on the entire volume $\mathcal{D}$, which is a three-dimensional grid. As it stands, it *does not* compare favorably with surface-based approaches, which operate on two-dimensional datasets, e.g, a surface mesh. The computational burden is alleviated by computed solutions for ϕ only in a narrow band that lies in vicinity of the zero set, as described in the literature [**?**, **?**, **?**]. In this particular application, the zero set moves less than one voxel unit from its original position (because of the constraints), and therefore the narrow band, usually about 9 voxels wide, remains constant throughout the fitting process.

Thus the complete, narrow-band algorithm is as follows:

1. Construct an initial solution $u^0_{i,j,k} = B_{i,j,k}$.
2. Find all of the grid points in $u^0_{i,j,k}$ that lie adjacent to one or more grid points of opposite sign. Call this set $\mathcal{A}_0$.
3. Find the set all of the grid points that are adjacent to $\mathcal{A}_0$ and denote it $\mathcal{A}_1$. Repeat this for $\mathcal{A}_2, \mathcal{A}_3, \ldots, \mathcal{A}_M$, to create a band that is $2M+1$ wide. The union of these sets, $\mathcal{A} = \cup_{i=0}^{M} \mathcal{A}_i$, is the *active set*.
4. For each $(i,j,k) \in \mathcal{A}$ calculate $\Delta u^n_{i,j,k}$ according to equation **??**.
5. For each $(i,j,k) \in \mathcal{A}$ update the value of $u^{n+1}_{i,j,k}$ according to equation **??**.
6. Find the find the average change for points in the active set:

$$c^n = \left(\frac{1}{|\mathcal{A}|}\sum_{\mathcal{A}} \left|u^{n+1}_{i,j,k} - u^n_{i,j,k}\right|^2\right)^{\frac{1}{2}}. \qquad (20)$$

7. If c^n is below some predefined threshold, then the algorithm is complete, otherwise increment n and go to step **??**.

6 Results

This section presents a series of results that demonstrate the effectiveness of the proposed algorithm. These results consist of shaded, 3D renderings of triangle meshes that are produced by marching cubes performed on the volumes that result from the proposed method. As an alternative to marching cubes, one could render this dataset with a direct method, e.g., using ray casting or splatting. The choice of marching cubes for rendering iso-surfaces of this and other datasets in this paper is not essential to the proposed method. The problems of aliasing cannot be removed by simply resampling or filtering along the coordinates of the viewing frustum, as one typically does when volume rendering. The particular problems with marching cubes, which are missing triangles at singularities and surface artifacts from acute triangles, are not the focus of this work and they do not have a significant impact on the results presented here.

Figures **??**a–b show the zero-sets of grey-scale and binary volumes of a cube. The grey-scale volume is the distance transform. The cubes shown are rotated 22.5° around each axis (in order to create significant aliasing artifacts in the binary version), and the edges of the cubes have a length of 50 grid units. Figure **??**c shows the zero-set of the solution to the constrained minimization problem with a stopping threshold of 0.002 using a narrow band of $M = 4$. These results show significant improvements in the aliasing, especially along the flat faces where the minimum-surface-area approach is most appropriate. On the corners and edges artifacts remain because the algorithm is trying to "stretch" the minimal surface across the constraints, which contain "jaggies". The algorithm converges rapidly, in about 20 iterations. Experiments show that the choice of band width and stopping threshold do not affect the results in any significant way—provided that the width is sufficiently large and the stopping threshold is sufficiently small.

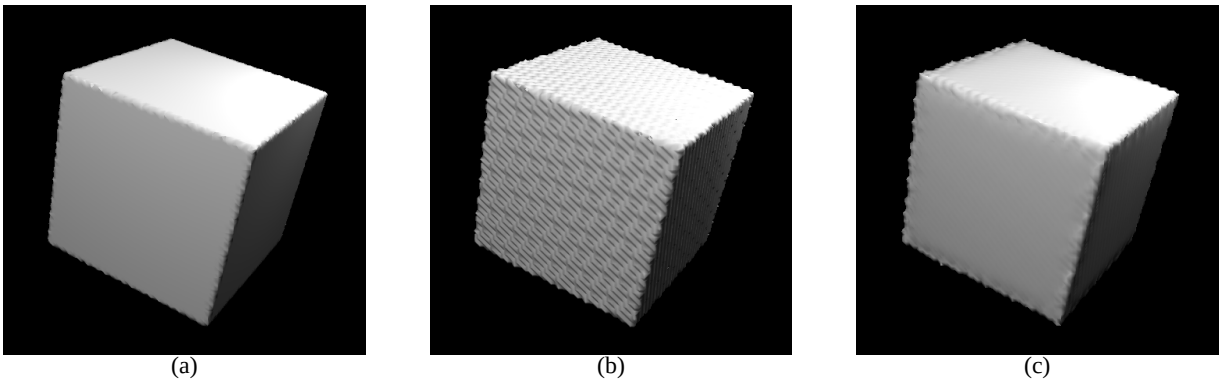
Figures **??**–**??** show a series of before (left) and after (right) iso-surface renderings. Generally the algorithm succeeds in reducing aliasing artifacts with a minimal distortion of the shapes. Notice that for some shapes, such as the low-resolution torus in figure **??** the aliasing is reduced, but only marginally so. That result demonstrates a fundamental limitation of the proposed algorithm; *the minimal surface criterion does not always get the solutions "close" to the ideal.* Instead the solution is stretching across the rather coarse features formed by the binary volumes. This is especially bad in cases such as a torus, which includes points for which one principle curvature is significantly greater than the other, causing the surface to "pucker" inward, leaving pronounced aliasing artifacts. On flat surfaces or those with higher resolution, the aliasing effects are virtually eliminated.

Figure **??**a shows a grey-scale slice of a CT image of the skull of an orangutan, which is $200 \times 200 \times 200$ voxels. Figure **??**b shows a thresholded version of that data, which provides a fairly complicated 3D, binary dataset for demonstrating the proposed algorithm.

Figures **??** and **??** show pairs of before and after images for the proposed algorithm. The reduction of aliasing artifacts is significant but not complete. However, the there is virtually no loss in detail or structure, as guaranteed by the fundamental properties of the algorithm.

Figure **??** shows renderings from that binary volume dataset using two different levels of Gaussian smoothing. The smoothing was implemented using the discrete approximation to small Gaussian kernels given by Lindeberg [**?**]. Such a low-pass filtering approach introduces a free parameter, and yet no single value is appropriate for keeping detail and achieving significantly reduced artifacts. At a standard deviation of $\sigma = 1.0$ Gaussian smoothing leaves significant aliasing artifacts, e.g., near the jaw, and yet distorts the shapes of dimples in the face and enlarges holes. Increasing σ by only 50%, to 1.5, yields very different results, with significant distortion and loss of detail.

As an aside, the mesh model that results from the *skull* dataset (shown on the right in Figure **??**) contains approximately 260,000 triangles, which are comprised of over 180,000 vertices. The binary image that gives rise to that dataset can be stored in row-major order and compressed using an adaptive Lempel-Ziv coding (the "compress" command in Unix), reducing the dataset to 63075 bytes.



(a) (b) (c)

Figure 1: a) An ideal grey-scale embedding of a cube (i.e., distance transform) results in a smooth, accurate iso-surface. b) A binary volume yields significant aliasing artifacts. c) The surface estimation from the binary data with $M = 4$ and a stopping threshold of 0.002 shows a quality that is comparable to the ideal.

This evidence suggests that using binary volumes, fairly complex objects could be transmitted at very low data rates. With the aide of the proposed algorithm, such models can be processed at the receiving end to reduce aliasing artifacts.

The computation times of the proposed algorithm are reasonable. Using the narrow band method, the computation times of surface estimation range from about 1 to 5 minutes on a SGI Octane with a MIPS 10000 185 MHz processor. Most of that time is spent scanning the entire volume to construct the active set. These times do not include the time required for I/O, which is comparable in magnitude to the times for processing.

7 Conclusions

Recovering grey-scale embeddings from binary data is an ill-posed problem. The problem can be solved by introducing a regularization that picks the feasible solution with the least surface area. This paper presented a technique for reducing aliasing artifacts in iso-surfaces of binary data based on that principle. The proposed method uses the input data as a set of constraints on a deformable surface that iteratively seeks to minimize its surface area. The deformable surface is embedded within the original input volume, and therefore the technique does not require the conversion from volumes to meshes or meshes back to volumes. The algorithm is based on first principles and is *complete* in the sense that it does not require parameter tuning. The proposed algorithm also guarantees a certain level of fidelity to the original data. The computation is limited to a narrow band around the zero-set of the data, and therefore computation times are comparable to parametric approaches.

Despite these promising results, there is room for improvement. The greatest shortcoming of this technique is the reliance on the minimal surface area solution, which is not always the "smoothest" solution and often shows residual aliasing artifacts. This is because minimizing surface area causes surfaces to stretch up against the constraints, which can result in a kind of "pucker" at places of high curvature. The solution to this is a better regularization term that goes beyond second-order information and seeks smooth surfaces rather than minimal surfaces. There is some promising work in the literature [**?**] that suggests this may be possible, but more work will be necessary in order invent efficient schemes for computing such higher-order flows on embedded surfaces.

Acknowledgments

Thanks to the Stanford University Computer Graphics Laboratory for making available the orangutan data. This work is supported by the Office of Naval Research grants N00014-97-0227 and N00014-00-0116.

REFERENCES

[1] D. Breen, S. Mauch, and R. Whitaker, "3D scan conversion of CSG models into distance volumes," in *Proceedings of the 1998 Symposium on Volume Visualization*, pp. 7–14, ACM SIGGRAPH, October 1998.

[2] S. Gibson, "Using distance maps for accurate surface representation in sampled volumes," in *1998 Symposium on Volume Graphics*, pp. 23–30, ACM SIGGRAPH, 1991.

[3] S. Wang and A. Kaufman, "Volume-sampled 3D modeling," *IEEE Computer Graphics and Aplications*, vol. 14, no. 5, pp. 26–32, 1994.

[4] S. Wang and A. Kaufman, "Volume-sampled voxelization of goemetric primitives," in *Proceedings of the 1993 Symposium on Volume Visualization*, pp. 78–84, ACM SIGGRAPH, October 1993.

[5] R. Machiraju and R. Yagel, "Reconstruction error characterization and control: A sampling theory approach," *IEEE Transactions on Vizualization and Computer Graphics*, vol. 2, pp. 364–378, December 1996.

[6] U. Tiede, T. Schiemann, and K. Hohne, "High quality rendering of attibuted volume data," in *IEEE Visualization 1998*, pp. 255–262, October 1998.

[7] J. Miller, D. Breen, W. Lorensen, R. O'Bara, and M. Wozny, "Geometrically deformed models: A method for extracting closed geometric models from volume data," *Computer Graphics (SIGGRAPH '91 Proceedings)*, vol. 25, pp. 217–226, July 1991.

[8] T. McInerey and D. Terzopoulos, "Deformable models in medical image analysis: A survey," *Medical Image Analysis*, vol. 1, no. 2, pp. 91–108, 1996.

[9] S. Gibson, "Using distance maps for accurate surface representation in sampled volumes," in *1998 Symposium on Volume Graphics*, pp. 23–30, ACM SIGGRAPH, 1991.

[10] S. Osher and J. Sethian, "Fronts propogating with curvature-

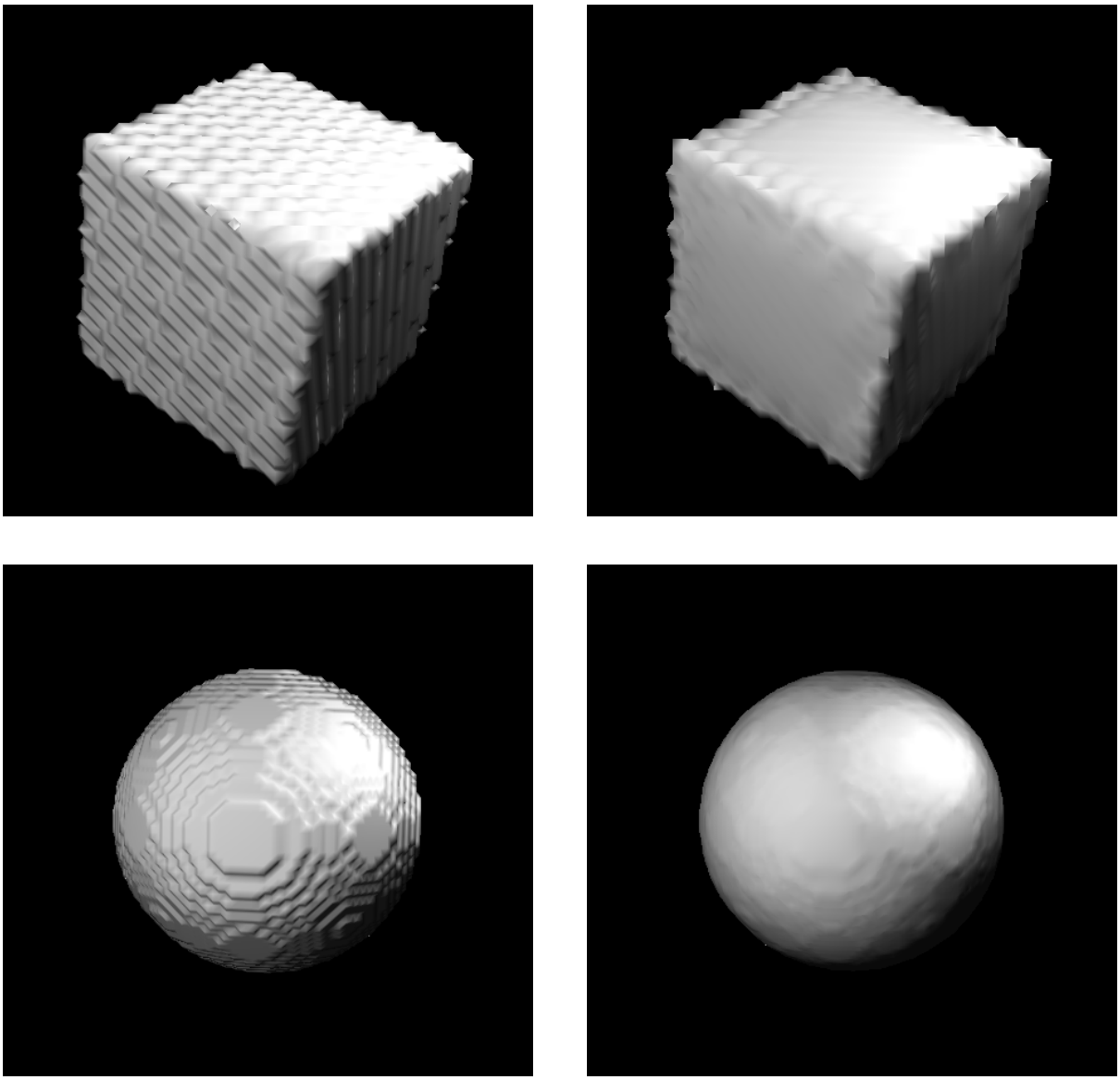


Figure 2: left) Binary input volumes. right) Results of surface estimation, top) low-resolution box, and bottom) sphere.

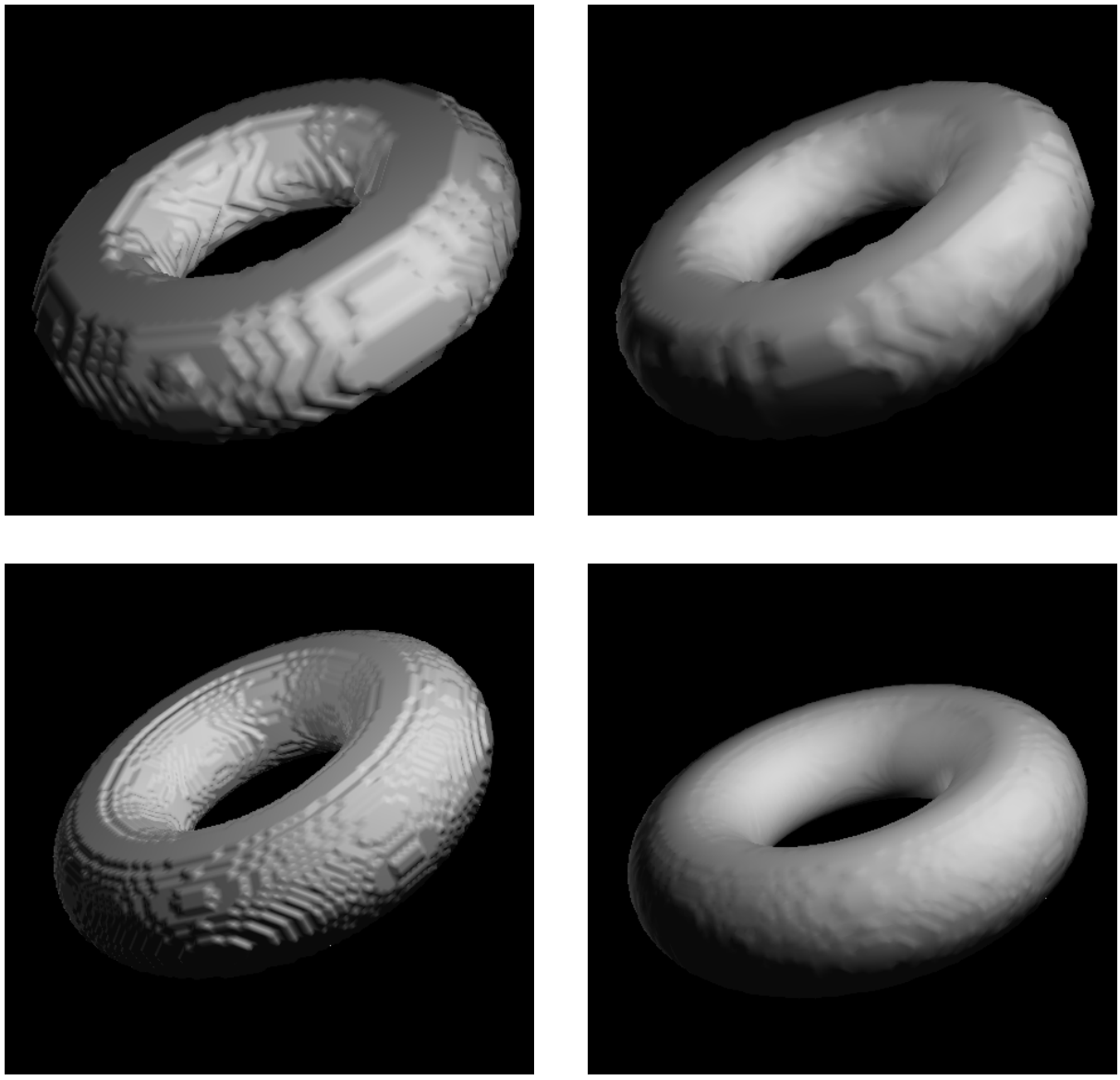
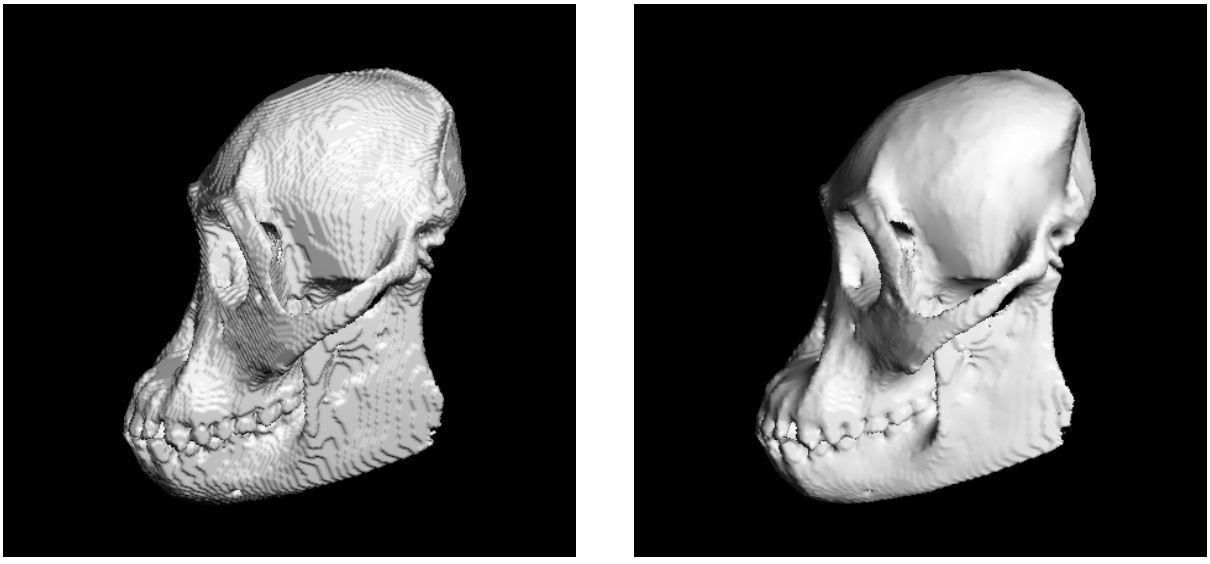


Figure 3: left) Binary input volumes. right) Results of surface estimation, top) low-resolution torus, and bottom) high-resolution torus.

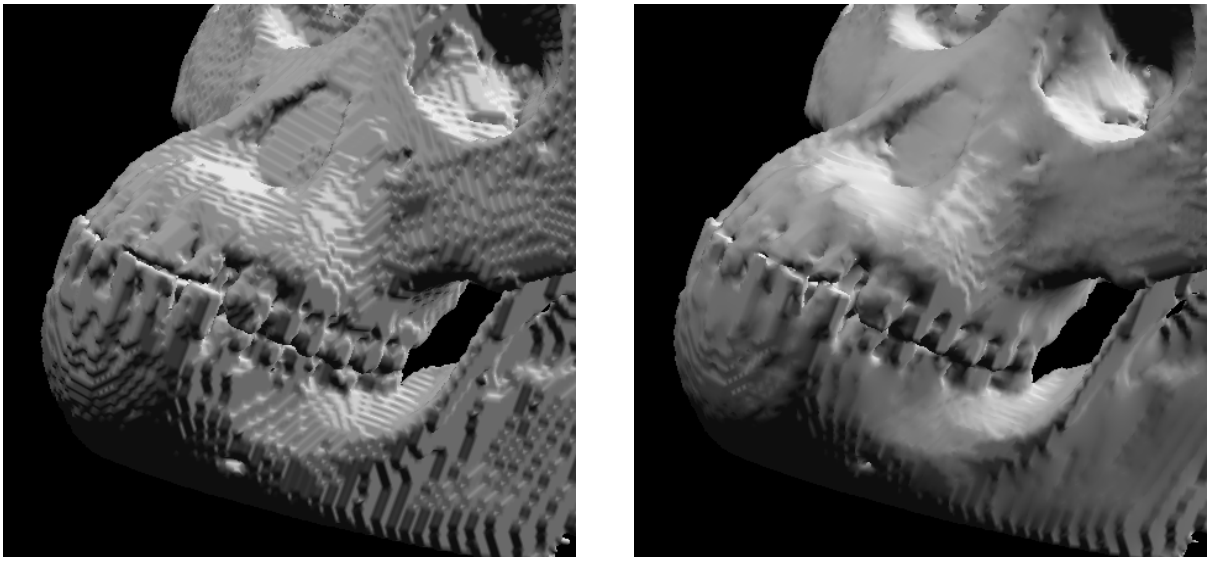
(a) (b)

Figure 4: (a) One slice of a $200 \times 200 \times 200$ CT dataset of an orangutan skull. (b) A thresholded version of that data provides a binary dataset—for demonstration purposes.



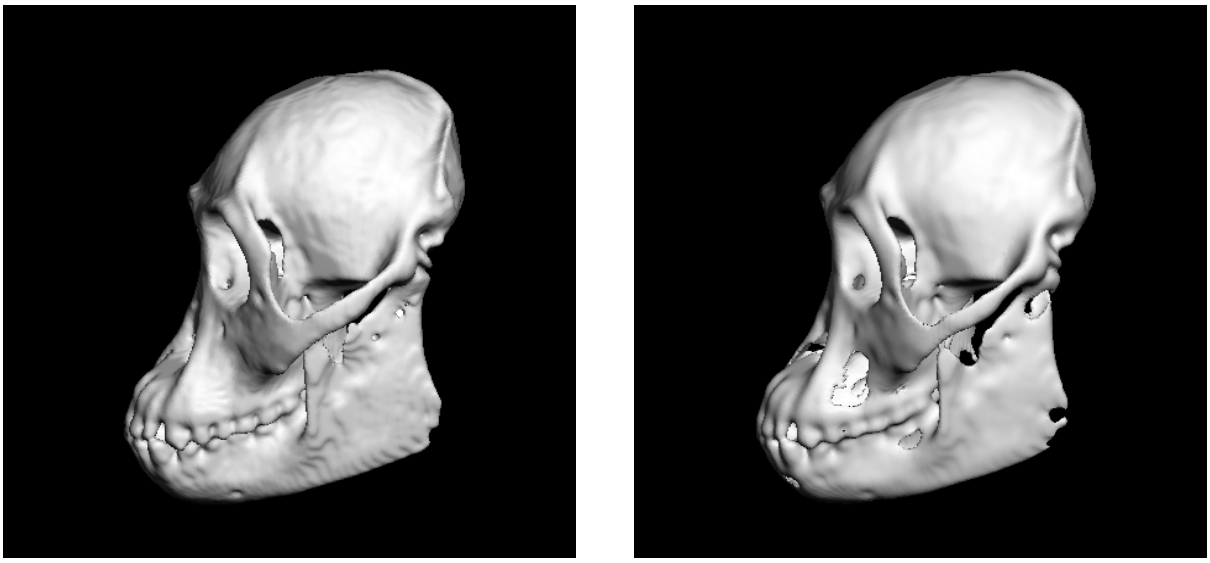
(a) (b)

Figure 5: Iso-surfaces of binary images from Figure **??** before (left) and after (right) processing.



(a) (b)

Figure 6: Iso-surfaces of binary images from Figure **??** before (left) and after (right) processing.



(a) (b)

Figure 7: (a) The binary dataset of figure **??** treated with Gaussian blurring of (a) $\sigma = 1.0$ and (a) $\sigma = 1.5$.

dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, pp. 12–49, 1988.

[11] J. Sethian, *Level Set Methods: Evolving Interfaces in Gometry, Fluid Mechanics, Computer Vision, and Material Sciences*. Cambridge University Press, 1996.

[12] R. Whitaker and D. Breen, "Level-set models for the deformation of solid objects," in *The Third International Workshop on Implicit Surfaces*, pp. 19–35, Eurographics, 1998.

[13] L. Cooper and D. Steinberg, *Introduction to Methods of Optimization*. New York: W.B. Saunders Company, 1970.

[14] D. Adalstein and J. Sethian, "A Fast Level Set Method for Propogating Interfaces," *Journal of Computational Physics*, pp. 269–277, 1995.

[15] R. T. Whitaker, "A Level-Set Approach to 3D Reconstruction From Range Data," *International Journal of Computer Vision*, vol. 29, pp. 203–231, Oct. 1998.

[16] T. Lindeberg, "A scale-space for descrete signals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 3, pp. 234–245, 1990.

Geometric Surface Processing via Normal Maps

Tolga Tasdizen	*Ross Whitaker*	*Paul Burchard*	*Stanley Osher*
Univ. of Utah	*Univ. of Utah*	*UCLA*	*UCLA*
tolga@cs.utah.edu	*whitaker@cs.utah.edu*	*burchard@pobox.com*	*sjo@math.ucla.edu*

University of Utah, School of Computing
Technical Report UUCS-02-03

University of California at Los Angeles
CAM report cam02-03

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

January 17, 2002

Abstract

The generalization of signal and image processing to surfaces entails filtering the normals of the surface, rather than filtering the positions of points on a mesh. Using a variational framework, smooth surfaces minimize the norm of the derivative of the surface normals—i.e. total curvature. Penalty functions on the surface normals are computed using geometry-based shape metrics and minimized using gradient descent. This produces a set of partial differential equations (PDE). In this paper, we introduce a novel framework for implementing geometric processing tools for surfaces using a two step algorithm: (i) operating on the normal map of a surface, and (ii) manipulating the surface to fit the processed normals. The computational approach uses level set surface models; therefore, the processing does not depend on any underlying parameterization. Iterating this two-step process, we can implement geometric fourth-order flows efficiently by solving a set of coupled second-order PDEs. This paper will demonstrate that the framework provides for a wide range of surface processing operations, including edge-preserving smoothing and high-boost filtering. Furthermore, the generality of the implementation makes it appropriate for very complex surface models, e.g. those constructed directly from measured data.

Chapter 1

Introduction

The fundamental principles of signal processing give rise to a wide range of useful tools for manipulating and transforming signals and images. The generalization of these principles to the processing of 3D surfaces has become an important problem in computer graphics, visualization, and vision. For instance, 3D range sensing technologies produce high resolution descriptions of objects, but they often suffer from noise. Medical imaging modalities such as MRI and CT scans produce large volumes of scalar or tensor measurements, but surfaces of interest must be extracted through some segmentation process or fitted directly to the measurements.

The state of the art in surface processing includes a number of very useful tools for processing meshes. However, to date there *is no general framework for geometric surface processing*. By *general* we mean two things. First, the framework should provide a broad variety of capabilities, including surface processing tools that resemble the state of the art in image processing algorithms. Second the framework should apply to a general class of surfaces. Users should be able to process complex surfaces of arbitrary topology, and obtain meaningful results with very little a priori knowledge about the shapes. By *geometric* we mean that output of surface processing algorithms should depend on surface shape and resolution, but should be independent of arbitrary decisions about the representation or parameterization.

This paper presents a framework that is based on the proposition that the natural generalization of image processing to surfaces is via the *surface normal vectors*. Thus, a smooth surface is one that has smoothly varying normals. In this light, the differences between surface processing and image processing are threefold. Normals live on a manifold (the surface) and cannot necessarily be processed using a flat metric, as is typically done with

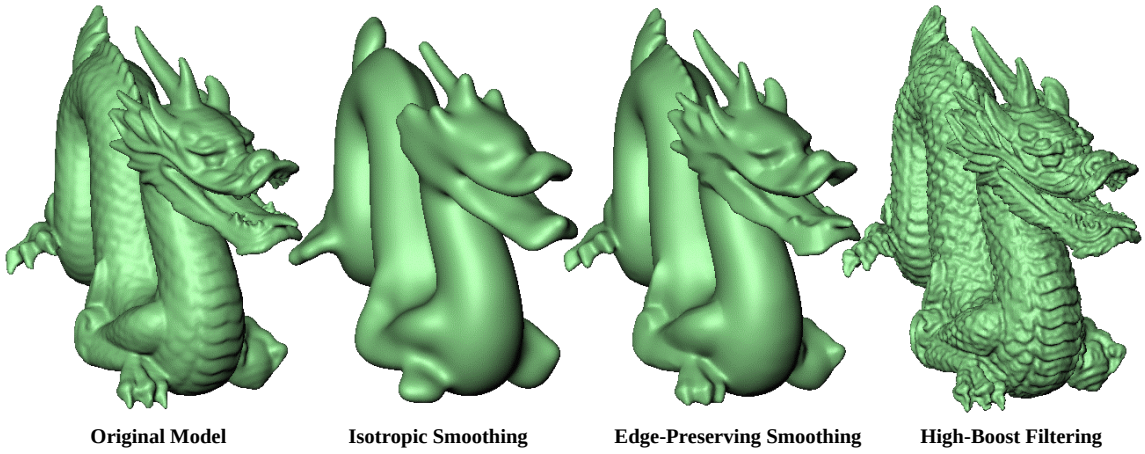


Figure 1.1: Surface processing examples.

images. Normals are vector valued and constrained to be unit length; the processing techniques must accommodate this. Normals are coupled with the surface shape, and thus the normals should drag the surface along as their values are modified during processing.

This paper presents an implementation that represents surfaces as the level sets of volumes and computes the processing of the normals and the deformation of the surfaces as solutions to a set of partial differential equations (PDE). This strategy enables us to achieve the "black box" behavior, which is reflected in the nature of the results. Results in this paper will show a level of complexity in the models and the processing algorithms that has not yet been demonstrated in the literature. This generality comes at the cost of significant computation time. However, the method is practical with state-of-the-art computers and is well-poised to benefit from parallel computing architectures, due to its reliance on local, iterative computations.

Figure 1 shows several results of processing a 3D surface model with different algorithms. These algorithms consist of smoothing, feature-preserving smoothing, and surface enhancement. All three processes are consistent mathematical generalizations of their image-processing counterparts. Note that all of the surfaces in this paper are represented volumetrically and rendered using the marching cubes algorithm [1].

In some applications, such as animation, models are manually generated by a designer and the parameterization is not arbitrary but is an important aspect of the geometric model. In these cases mesh-based processing methods offer a powerful set of tools, such as hierarchical editing [2], which are not yet possible with the proposed representation. However, in other applications, such as 3D segmentation and surface reconstruction [3, 4], the pro-

cessing is data driven, and surfaces can deform quite far from their initial shapes and even change topology. Furthermore, when considering processes other than isotropic smoothing, such as nonlinear smoothing or high-boost filtering, the creation or sharpening of small features can exhibit noticeable effects of the mesh topology—features that are aligned with the mesh are treated differently than those that are not. The techniques presented in this paper offer a new set of capabilities that are especially interesting when processing measured data—as are all of the examples shown in this paper.

The specific contributions of this paper are:

- a novel framework for geometric processing of surfaces that relies on surface normals;
- a numerical method for solving fourth-order level set equations in two simpler steps, thereby avoiding the explicit computation of unstable high-order derivatives; and
- examples of three geometric processing algorithms with applications to data sets that are more complex than those previously demonstrated in the literature.

Chapter 2

Related Work

The majority of surface processing research has been in the context of *surface fairing* with the motivation of smoothing surface models to create aesthetically pleasing surfaces using triangulated meshes [5, 6, 7, 8]. Surface fairing typically operate by minimizing a fairness or penalty function that favors smooth surfaces [9, 10, 11, 5]. Fairness functionals can depend on the geometry of the surface or the parameterization. Geometric functionals make use of invariants such as principal curvatures, which are parameterization independent, intrinsic properties of the surface. Therefore, geometric approaches produce results that are not affected by arbitrary decisions about the parameterization; however, geometric invariants are nonlinear functions of surface derivatives that are computationally expensive to evaluate. Simpler parameterization dependent functionals are linear approximations to geometric invariants. Such functionals can be equivalent to geometric invariants when the surface parameterization is *isometric* or they can be poor approximations when the parameterization is irregular and non-differentiable. An *isometric* surface parameterization requires the two parameter coordinate axis to be orthogonal and arc-length parameterized. In the context of surface fairing with meshes these concepts are also referred to as geometric and parameterization smoothness [7] or outer and inner fairness [12].

One way to smooth a surface is to incrementally reduce its surface area. This can be accomplished by mean curvature flow (MCF) at every point S:

$$\frac{\partial S}{\partial t} = \mathrm{H}\vec{\mathrm{N}} \tag{2.1}$$

where H is the mean curvature of the surface, $\vec{N}$ is the surface normal, and t is the time evolution of the surface shape. For parameterized surfaces the surface area translates to the

membrane energy functional

$$\int_\Omega X_u^2 + X_v^2 \, du \, dv \tag{2.2}$$

where $X(u,v)$ and Ω are the surface parameterization and its domain, respectively. For an isometric parameterization $X_u^2 + X_v^2 = 1$; therefore, (2.2) reduces to surface area. However, for larger and smaller $X_u^2 + X_v^2$, the approximation to surface area is distorted proportionally. The variational derivative of (2.2) is the Laplacian,

$$\Delta X = X_{uu} + X_{vv}, \tag{2.3}$$

which is equivalent to mean curvature in the isometric case. Laplacian, or mean curvature flow, is closely tied to Gaussian filtering—a standard method of smoothing images. Clarenz *et al.* [13] propose a meshed-based intrinsic flow that incorporates a weighted sum principle curvatures that depends on the local surface shape.

A second-order penalty function is *total curvature*

$$\int_S \kappa_1^2 + \kappa_2^2 \, dS \tag{2.4}$$

which has been shown to deform surfaces into spheres [14]. Total curvature is a geometric (invariant) property of the surface. The mesh fairing approach of [5] which minimizes (2.4) involves fitting local polynomial basis functions to local neighborhoods for the computation of total curvature. These polynomial basis functions range from full quadratic polynomials to constrained quadratics and planar approximations. Depending on the complexity of the local neighborhood, the algorithm must choose, at each location, which basis to employ. Ambiguities result at locations where multiple basis provide equally good representations. In [8] the authors search directly for an intrinsic PDE that produces fair surfaces instead of deriving the PDEs from a variational framework. They propose the Laplacian of mean curvature $\Delta_B \mathrm{H} = 0$ for meshes where Δ_B is the Laplace-Beltrami operator, i.e. the Laplacian for parameterized surfaces. Their approach is not sufficiently general to satisfy the goals of this paper, but is closely related to the proposed method. We will discuss it further in Sec. 3.

If we penalize the parameterization (i.e. non-geometric), equation (2.4) becomes the thin plate energy functional

$$\int_\Omega X_{uu}^2 + 2X_{uv}^2 + X_{vv}^2 \, du \, dv \tag{2.5}$$

where X and Ω are as defined for (2.2). Thin plate energy was used in [10] for surface fairing. The variational derivative of (2.5) is the biharmonic operator, which is linear:

$$\Delta^2 X = X_{uuuu} + 2X_{uuvv} + X_{vvvv}. \tag{2.6}$$

These linear energy functionals underly the signal processing approach to surface fairing pioneered in [6], who derived the natural vibration frequencies of a surface from the Laplacian operator. Taubin observes that Gaussian filtering causes shrinkage. He eliminates this problem by designing a low pass filter using a weighted average of the Laplacian and the biharmonic operator. The weights have to be fine-tuned to obtain the non-shrinking property. Analyzed in the frequency domain, this low-pass filter can be seen as a Gaussian smoothing shrinking step followed by an unshrinking step. Indeed, any polynomial transfer function in the frequency domain can be implemented with this method [15]. [16] describe a related approach in which surfaces are smoothed by simultaneously solving the membrane (2.2) and thin plate (2.5) energy functionals.

The signal processing approaches use the umbrella operator which is a discretization of the Laplacian. The edge lengths connecting the nodes of the mesh and the angles between adjacent edges around a node, also known as face angles, introduce parameterization dependencies. By setting the edge weights in the umbrella operator to the reciprocal of the edge length, the dependency on edge length can be removed [6], but the dependency on the face angles remain. A scale dependent intrinsic umbrella operator is defined in [7] that removes both dependencies. The time steps in explicitly integrating a scale dependent umbrella operator are proportional to the square of the shortest edge length. Desbrun *et al.* overcome this limitation by introducing an implicit integration scheme. Nevertheless, the weights for the umbrella operator must be recomputed at each iteration to maintain its intrinsic property. A non-uniform relaxation operator is introduced in [2] to minimize a locally weighted quadratic energy of second order differences.

Moreton and Sequin [9] propose a geometric fairness functional that penalizes the variation of principle curvatures—a third-order, geometric penalty function (corresponding to a *sixth-order* variational derivative), which requires very large computation times. The analysis and implementation of general penalty functions above second order remains an open problem, which is beyond the scope of this paper. Evidence in this paper and elsewhere [7, 8] suggests that fourth-order geometric flows form a sufficient foundation for a general, geometric surface processing system.

The work in this paper is also related to that of Chopp & Sethian [17], who derive the intrinsic Laplacian of curvature for an implicit curve, and solve the resulting fourth-order nonlinear PDE. However, they argue that the numerical methods used to solve second order flows are not practical, because they lack long term stability. They propose several new numerical schemes, but none are found to be completely satisfactory due to their slow computation and inability to handle singularities. One of the results of this paper is to solve this equation more effectively and to demonstrate that this is only one example of a more general class of useful surface processing techniques. Joint interpolation of vector fields and gray level functions was used for succesfully filling-in missing parts of images in [18].

Chapter 3

Geometric Surface Processing

One of the underlying strategies of this paper is to use *geometric surface processing*, where the output of the process depends only on the shape of the input surface, and does not contain artifacts from the underlying parameterization. The motivation for this strategy is discussed in detail in [12], where the influence of the mesh parameterization on surface fairing results is clearly shown, and higher-order geometric flows, such as the intrinsic Laplacian of curvature, are proposed as the solution.

As an illustration of the importance of higher-order geometric processing, consider the results in Fig. 3.1, which demonstrates the differences between processing surfaces with mean curvature flow (MCF) and intrinsic Laplacian of mean curvature flow (ILMCF). The amount of smoothing for MCF and ILMCF was chosen to be qualitatively similar, and yet important differences can be observed on the smaller features of the original model. MCF has shortened the horns of the original model, and yet they remain sharp—not a desirable behavior for a "smoothing" process. This behavior for MCF is well documented as a pinching off of cylindrical objects and is expected from the variational point of view: MCF minimizes surface area and therefore will quickly eliminate smaller parts of a model. Some authors [19] have proposed volume preserving forms of second-order flows, but these processes compensate by enlarging *the object as a whole*, which exhibits, qualitatively, the same behavior on small features. Intrinsic Laplacian of mean curvature flow, in Fig. 3.1, preserves the structure of these features much better *while* smoothing them.

An alternative to solving a fourth-order equation directly is to decouple it into a pair of second-order equations. For instance, a two-step solution to ILMCF for meshes is proposed in [8]. However, this approach works only for meshes, and relies on analytic properties of the steady-state solutions, $\Delta H = 0$, by fitting surface primitives that have those properties.

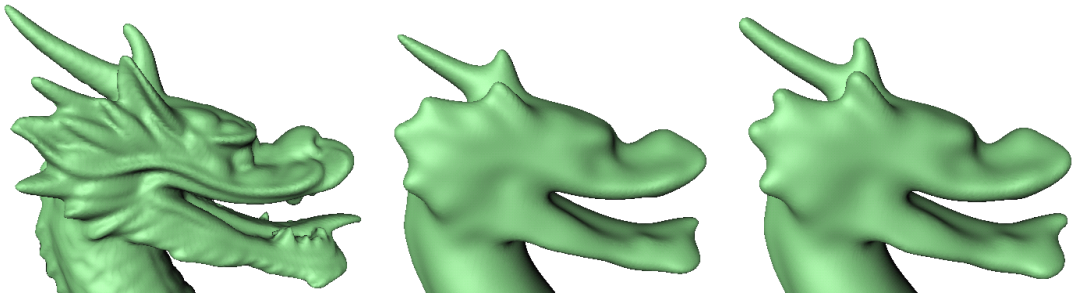


Figure 3.1: Second- and fourth-order surface smoothing. From left to right: Original model, mean curvature flow, and intrinsic Laplacian of mean curvature flow.

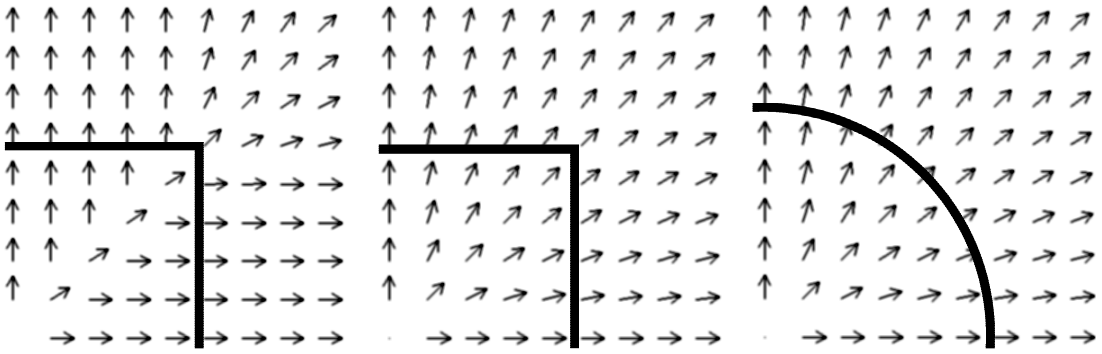


Figure 3.2: Shown here in 2D, the process begins with a shape and constructs a normal map from the distance transform (left), modifies the normal map according to a PDE derived from a penalty function (center), and re-fits the shape to the normal map (right).

Thus, the formalism does not generalize well to applications, such as surface reconstruction, where the solution is a combination of measured data and a fourth-order smoothing term. Also, it does not apply to other types of smoothing processes, such as those that minimize nonlinear feature-preserving penalties.

In [18], the authors penalize the smoothness of a vector field while simultaneously forcing the gradient directions of a gray scale image to closely match the vector field. The penalty function on the normal field is proportional to the divergence of the normal vectors. This forms a high-order interpolation function, which is shown to be useful for image inpainting—recovering missing patches of data in 2D images. This strategy of simultaneously penalizing the divergence of a normal field and the mismatch of this field with the image gradient is closely related to the total curvature penalty function used in this paper. The formulation proposed in this paper emphasizes the processing of normals on an arbitrary surface manifold (rather than the flat geometry of an image), with an explicit relationship to fourth-order surface flows. Furthermore, this paper establishes new directions for surface flows— toward edge-preserving surface smoothing and feature enhancement.

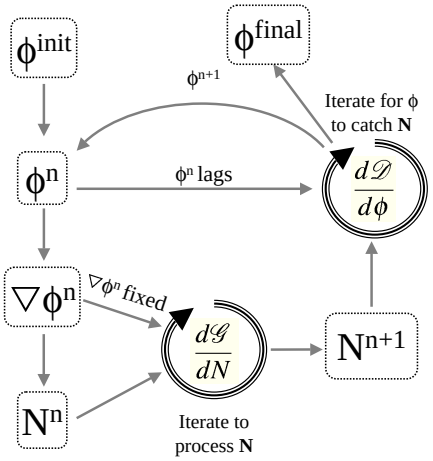


Figure 3.3: Flow chart

In this section, we will generalize [17] to the intrinsic Laplacian of mean curvature for level set surfaces, and introduce a method for breaking it into two simpler PDEs. This pair of equations is solved by allowing the surface shape to lag the normals as they are filtered and then catch up by a separate process. Figure 3.2 shows this three step process graphically in 2D—shapes give rise to normal maps, which, when filtered, give rise to new shapes. In the limit, this approach is equivalent to solving the full-blown, intrinsic fourth-order flow, but it generalizes to a wide range of processes and makes no assumptions about the shapes of the solutions. In Sec. 4, we show results for several different kinds of surface filters. The appendix describes a numerically stable discrete solver for the system of equations.

3.1 Level set methods

The remainder of this paper addresses the higher-order geometry of implicit surfaces, i.e. level sets. Even though very good surface fairing results have been obtained with meshes, there are some drawbacks to using meshes and other parametric models for purposes other than simple smoothing (i.e. low pass filtering). We believe that level set methods [20, 21] are better suited for a wide range of surface-processing for the following reasons.

1. Some surface processes, such as anisotropic diffusion and high-boost filtering, have the capability of introducing new features—a fundamental difference from smoothing. Meshes do not form discontinuities well unless the edges of the face triangles coincide with the edges of the surface.

2. Anything more than a modest amount of smoothing is likely to evolve the surface far from its original shape. This requires the creation and deletion of faces in meshes to maintain the original resolution. Implicit surfaces do not exhibit this problem.
3. For surface reconstruction, such as from range imagery or tomographic data, the evolving surface can undergo topological changes [3, 4]. Surface meshes do not (readily) allow topological changes, whereas level set surfaces do.

To facilitate the discussion, we use the Einstein notation convention, where the subscripts indicate tensor indices, and repeated subscripts within a product represent a summation over the index (across the dimensions of the underlying space). Furthermore, we use the convention that subscripts on quantities represent derivatives, except where they are in parenthesis, in which case they refer to a vector-valued variable. Thus, ϕ_i is the gradient vector of a scalar quantity $\phi : \mathbb{R}^n \mapsto \mathbb{R}$. The Hessian is ϕ_{ij}, and the Laplacian is ϕ_{ii}. A vector field is $v_{(i)}$, where $v : \mathbb{R}^n \mapsto \mathbb{R}^n$, and the divergence of that field is $v_{(i)i}$. Scalar operators, such as differentials behave in the usual way. Thus, gradient magnitude is $|\phi_i| = \sqrt{\phi_i \phi_i}$ and the differential for a coordinate system is $dx_{(i)} = dx_1 dx_2 \ldots dx_n$.

Level set surface models rely on the notion of a regular surface, which is a collection of 3D points, $\mathscr{S}$, with a topology that allows each point to be modeled *locally* as a function of two variables. We can describe the deformation of such a surface using the 3D velocity of each of its constituent points, i.e., $\partial s_{(i)}(t)/\partial t$ for all $s_{(i)} \in \mathscr{S}$. If we represent the surface implicitly at each time t, then

$$\mathscr{S} = \left\{ s_{(i)}(t) \mid \phi\left(s_{(i)}(t), t\right) = 0 \right\}. \tag{3.1}$$

Notice that surfaces defined in this way divide a volume into two parts: inside ($\phi > 0$) and outside ($\phi < 0$). It is common to choose ϕ to be the signed distance transform of $\mathscr{S}$, or an approximation thereof.

The surface remains a level set of ϕ over time, and thus taking the total derivative with respect to time (using the chain rule) gives

$$\frac{\partial \phi}{\partial t} = -\phi_j \frac{\partial s_{(j)}}{\partial t} \tag{3.2}$$

Notice that ϕ_j is proportional to the surface normal, and thus $\partial s_{(j)}/\partial t$ affects ϕ only in the direction of the surface normal— surface motion in any other direction is merely a change in the parameterization. Using this framework, the PDE on ϕ that describes the motion of a surface by mean curvature, as in (2.1), is

$$\frac{\partial \phi}{\partial t} = \phi_{ii} - \frac{\phi_{ij}\phi_i\phi_j}{\phi_k\phi_k}. \tag{3.3}$$

The following sections give the mathematical description of the process outlined in Fig. 3.3. First we will derive the energy functional for minimizing the total curvature of a normal map. The first variation of the total curvature results in a second-order PDE on the normals. This process is denoted in Fig. 3.3 as the $d\mathscr{G}/d\mathbf{N}$ loop. Next, in Sec. 3.3, we show another energy functional for fitting a surface to an evolved normal map, resulting in a second-order PDE on ϕ. This is the $d\mathscr{D}/d\phi$ loop in Fig. 3.3. Finally, we will show that the overall process of simultaneously solving these two PDEs as shown in Fig. 3.3 is equivalent to ILMCF. This establishes the mathematical foundation of the proposed method.

3.2 Intrinsic Laplacian of mean curvature flow for normal maps

When using implicit representations one must account for the fact that derivatives of functions defined on the surface are computed by projecting their 3D derivatives onto the tangent plane of the surface. Let $N_{(i)} : \mathbb{R}^3 \to S^3$ be the normal map, which is a field of normals that are everywhere perpendicular to the family of embedded isosurfaces of ϕ—thus $N_{(i)} = \phi_i / \sqrt{\phi_k \phi_k}$. The 3×3 projection matrix for the implicit surface normal is $P_{(ij)} = N_{(i)} N_{(j)}$, and $P_{(ij)} V_{(i)}$ returns the projection of $V_{(i)}$ onto $N_{(i)}$. Let $I_{(ij)}$ be the identity matrix. Then the projection onto the plane that is perpendicular to the vector field $N_{(i)}$ is the *tangent projection operator*, $T_{(ij)} = I_{(ij)} - P_{(ij)}$. Under typical circumstances the normal map $N_{(i)}$ is derived from the gradient of ϕ, and $T_{(ij)}$ projects vectors onto the tangent planes of the level sets of ϕ. However, the computational strategy we are proposing allows ϕ to lag the normal map. Therefore, the tangent projection operators differ, and we use $T^{\phi}_{(ij)}$ to denote projections onto the tangent planes of the level sets of ϕ and $T^{N}_{(ij)}$ to denote projections onto planes perpendicular to the normal map.

The shape matrix [22] of a surface describes its curvature independent of the parameterization. The shape matrix of an implicit surface is obtained by differentiating the normal map and projecting that derivative onto the tangent plane of the surface. The Euclidean norm of the shape matrix is the total curvature, κ. Thus

$$\kappa^2 = \left|\left| N_{(i)j} T^{\phi}_{(jk)} \right|\right|^2. \tag{3.4}$$

If $N_{(i)}$ is derived directly from ϕ, this gives

$$\kappa^2 = \left|\left| T^{\phi}_{(il)} \phi_{lj} T^{\phi}_{(jk)} \right|\right|^2. \tag{3.5}$$

The strategy is to treat these expressions as penalty functions, and develop surface processing methods by solving the PDEs that result from the first variation of surface integrals over these penalty functions. Notice, that if we take the first variation of (3.5) we obtain a fourth-order PDE on ϕ, but if we take the first variation of (3.4), with respect to $N_{(i)}$, allowing ϕ to remain fixed, we obtain a second-order PDE on $N_{(i)}$. To see the first variation of (3.4) we re-express the norm, using the identity $||A||^2 = \text{Trace}[\text{A}^\text{T}\text{A}]$. This gives

$$\kappa^2 = \left|\left|N_{(i)j}\right|\right|^2 - \frac{\left|\left|N_{(i)j}\phi_j\right|\right|^2}{\phi_k\phi_k}, \tag{3.6}$$

and thus the penalty function for the total surface curvature is

$$\mathscr{G}\left(N_{(i)}\right) = \int_{\mathscr{S}} \left[\left|\left|N_{(i)j}\right|\right|^2 - \frac{\left|\left|N_{(i)j}\phi_j\right|\right|^2}{\phi_k\phi_k}\right] dx_{(m)}. \tag{3.7}$$

As we process the normal map $N_{(i)}$, letting ϕ lag, we must ensure that it maintains the unit length constraint, $N_{(i)}N_{(i)} = 1$. This is expressed in the penalty function using Lagrange multipliers. The constrained penalty is

$$\mathscr{G}\left(N_{(i)}\right) + \int_{\mathscr{S}} \lambda(x_{(l)})\left(N_{(k)}N_{(k)} - 1\right) dx_{(l)}, \tag{3.8}$$

where $\lambda(x_{(l)})$ is the Lagrange multiplier at $x_{(l)}$. Using the first variation and solving for λ introduces a projection operator $T^N_{(ij)}$ on the first variation of $\mathscr{G}$, which keeps $N_{(i)}$ unit length. Thus the first variation of (3.8) with respect to $N_{(i)}$ is

$$-T^N_{(ij)}\frac{d\mathscr{G}}{dN_{(j)}} = -2T^N_{(ij)}\left[N_{(j)k} - \frac{N_{(j)l}\phi_l\phi_k}{\phi_m\phi_m}\right]_k. \tag{3.9}$$

A gradient descent on this metric $\partial N_{(i)}/\partial t = -d\mathscr{G}/dN_{(i)}$, results in a PDE that smoothes the normal map by minimizing total curvature while maintaining the unit normal constraint. Notice that this is precisely the same as solving the constrained diffusion equation on $N_{(i)}$ using the method of solving PDEs on implicit manifolds described in [23]. This derivation is what we consider the *base case*—subsequent sections of this paper will introduce other processes on the normal maps.

3.3 Surface evolution via normal maps

We have shown how to evolve the normals to minimize total squared curvature; however, the final goal is the reconstruction of the surface which requires evolving ϕ, rather than the

normal map. Hence, the next step is to relate the evolution of ϕ to the evolution of $N_{(i)}$. Suppose that we are given the normal map $N_{(i)}$ to some set of surfaces, but not necessarily level sets of ϕ—as is the case if we filter $N_{(i)}$ and let ϕ lag. We can manipulate ϕ so that it *catches* up to $N_{(i)}$ by minimizing a penalty function that quantifies the discrepancy. This penalty function is

$$\mathscr{D}(\phi) = \int_U \left[\sqrt{\phi_i \phi_i} - \phi_i N_{(i)} \right] dx_{(j)}, \tag{3.10}$$

where $U \subset \mathbb{R}^3$ is the domain of ϕ. A gradient descent on ϕ that minimizes this penalty function is

$$\frac{\partial \phi}{\partial t} = -\frac{d\mathscr{D}}{d\phi} = ||\phi_k|| \left[\left(\frac{\phi_j}{\phi_m \phi_m} \right)_j - N_{(j)j} \right] = |\nabla \phi| \left[\mathrm{H}^{\phi} - \mathrm{H}^{\mathrm{N}} \right] \tag{3.11}$$

where H^{ϕ} is the mean curvature of the level set surface and H^{N} is the induced curvature of the normal map. Thus, the surface moves as the difference between its own curvature and that of the normal field. The factor of $|\nabla \phi|$, which is typical with level set formulations, comes from the fact that we are manipulating the shape of the level set, which is embedded in ϕ, as in (3.2).

We propose to solve fourth-order flows on the level sets of ϕ by a splitting strategy, which entails processing the normals and allowing ϕ to lag and then catch up later, in a separate process. This is only correct if we know that in the limit, as the lag becomes very small, we are solving the full fourth-order PDE. To show this we analyze one iteration of the main loop in Fig. 3.3. Before processing the normals, they are derived from ϕ^n, and we have $N^n_{(i)} = \phi^n_i / \sqrt{\phi^n_j \phi^n_j}$. Evolving the normal map once according to (3.9) for a small amount of time dt gives

$$N^{n+1}_{(i)} = N^n_{(i)} - T^N_{(ij)} \frac{d\mathscr{G}}{dN_{(j)}} dt. \tag{3.12}$$

If we immediately apply (3.11) to fit ϕ to this new normal map

$$\frac{\partial \phi}{\partial t} = \| \phi^n_k \| \left(\mathrm{H}^{\phi} - \left[\mathrm{N}^{\mathrm{n}}_{(\mathrm{i})} - \mathrm{T}^{\mathrm{N}}_{(\mathrm{ij})} \frac{\mathrm{d}\mathscr{G}}{\mathrm{dN}_{(\mathrm{j})}} \mathrm{dt} \right]_{\mathrm{i}} \right). \tag{3.13}$$

Because $N^n_{(i)}$ is derived directly from ϕ^n, we have $N^n_{(i)i} = \mathrm{H}^{\phi}$, which gives the expression for changes in ϕ in order to make up this infinitesimal lag:

$$\frac{\partial \phi}{\partial t} = - \| \phi^n_k \| \left[T^N_{(ij)} \frac{d\mathscr{G}}{dN_{(j)}} \right]_i. \tag{3.14}$$

We can express the penalty function $\mathcal{G}$ in terms of either $N_{(i)}$ or ϕ, and take the first variation with respect to either of these quantities. The relationship between $d\mathcal{G}/dN_{(i)}$ and $d\mathcal{G}/d\phi$ is established by integrating (by parts)—it is

$$\frac{d\mathcal{G}}{d\phi} = -\left[T_{ij}^{\phi} \frac{d\mathcal{G}}{dN_{(j)}} \right]_i , \tag{3.15}$$

where $T_{ij}^{\phi} = T_{ij}^{N}$ as $dt \to 0$. Here we have assumed that ϕ is a signed distance function; therefore $\mid \nabla\phi \mid = 1$. In our level set implementation, ϕ is maintained as an approximation to the signed distance transform, see [4]. The update on ϕ after it lags $N_{(i)}$ by some small amount is actually

$$\frac{\partial\phi}{\partial t} = \parallel \phi_k \parallel \frac{d\mathcal{G}}{d\phi}, \tag{3.16}$$

which is a gradient descent, of the level sets of ϕ, on $\mathcal{G}$— the total curvature of the surface.

This analysis shows that the strategy depicted in Figs. 3.2 and 3.3 is a valid mechanism for implementing the base-case, which is ILMCF on implicit, level set surfaces. However, this strategy has broader implications. Sec. 4.2 will show that other choices of $\mathcal{G}$ (there are many options from the image processing literature) will produce different kinds of PDE-based surface processing algorithms. Furthermore, Sec. 4.2.1 will demonstrate that the general strategy of processing normals separately from surfaces, in a two phase process, offers a set of useful tools that go beyond the strict variational framework.

Chapter 4

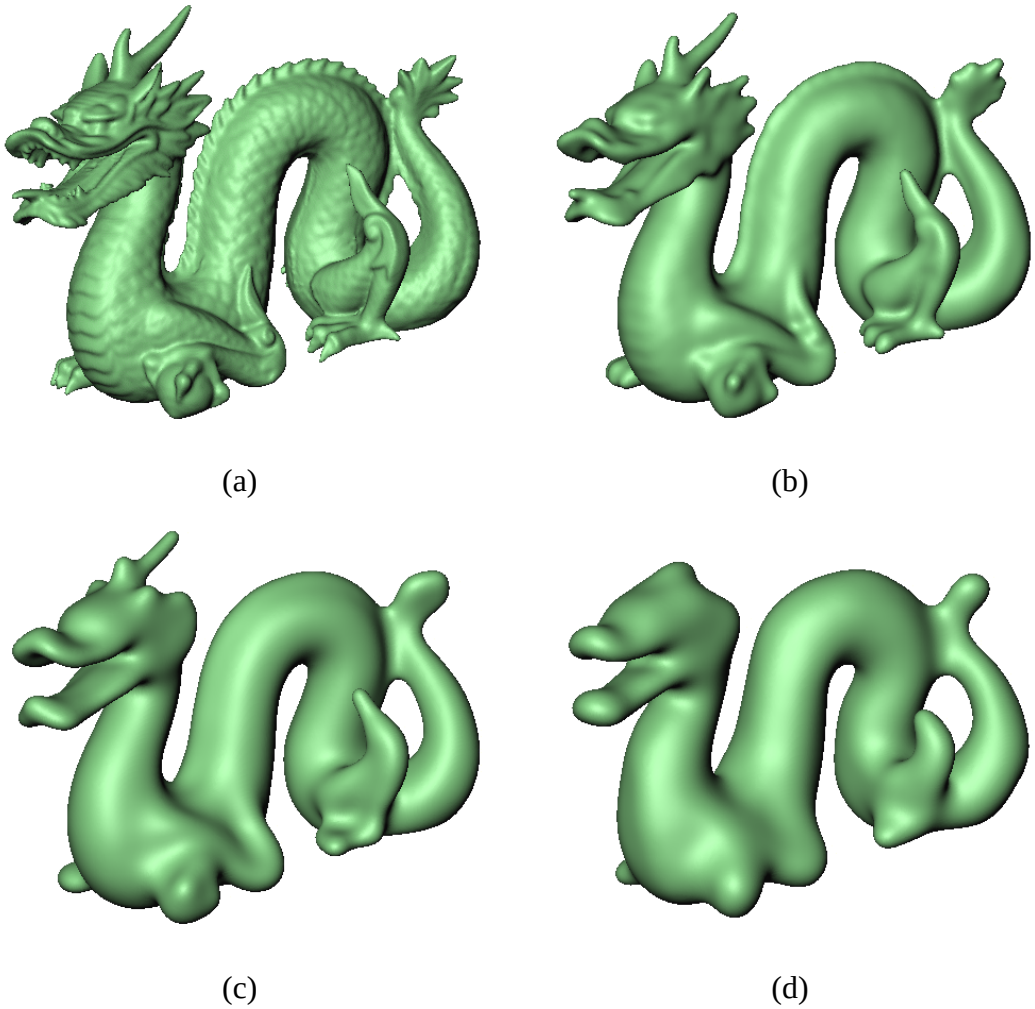
Applications

4.1 Isotropic Diffusion

We have discussed ILMCF in detail in Sec. 3. Here we will present some results of this diffusion process. Figure 4.1 shows a model at various stages of the diffusion process. Two iterations of the main processing loop in Fig. 3.3 are enough to remove the smallest scale features from the model. Later iterations start smoothing the global shape of the model. The model shown in this example consists of a $356 \times 161 \times 251$ volume. The computation time required for one iteration of the main processing loop for this model is around 30 minutes on a *1.7 Ghz Intel processor*. The models used in the examples in the rest of this paper approximately have the same level of complexity.

4.2 Anisotropic Diffusion

Minimizing the total squared curvature of a surface works well for smoothing surfaces and eliminating noise, but it also deforms or removes important features. This type of smoothing is called isotropic because it corresponds to solving the heat equation on the normal map with a constant, scalar conduction coefficient. This is equivalent to a convolution of the surface normals with a Gaussian kernel that is isotropic (using a metric that is flat on the surface). Isotropic diffusion is not particularly effective if the goal is to de-noise a surface that has an underlying structure with fine features. This scenario is common when extracting surfaces from 3D imaging modalities, such as magnetic resonance imaging (MRI),



(a) (b)

(c) (d)

Figure 4.1: Various stages of isotropic diffusion: (a) original model, (b) after 2 iterations, (c) after 8 iterations, and (d) after 25 iterations.

in which the measurements are inherently noisy. Figure 4.2(a) is an example of the skin surface, which was extracted, via isosurfacing, from an MRI data set. Notice that the roughness of the skin is noise, an artifact of the measurement process. This model is also quite complex because, despite our best efforts to avoid it, the isosurfaces include many convoluted passages up in the sinuses and around the neck. As an indication of this complexity, consider that marching cubes produces 543,000 triangles from this volume. This is over 10 times the number of faces in many of the "standard" models used to demonstrate surface processing algorithms [2, 7]. The strategy in this paper is to treat such surfaces as they are measured, in their volumetric representation, rather than process this data indirectly via a simplified, fitted surface mesh [24].

Isotropic diffusion, shown in Fig. 4.2(b), is marginally effective for de-noising the head surface. Notice that the sharp edges around the eyes, nose, lips and ears are lost in this process. The problem of preserving features while smoothing away noise has been studied extensively in computer vision. Anisotropic diffusion introduced in [25] has been very successful in dealing with this problem in a wide range of images. Perona & Malik proposed to replace Laplacian smoothing, which is equivalent to the heat equation $\partial I/\partial t = \nabla \cdot \nabla I$, with a nonlinear PDE

$$\partial I/\partial t = \nabla \cdot [g(\| \nabla I \|) \nabla I], \tag{4.1}$$

where I is generally the grey-level image, and $g(\cdot)$ is the edge stopping function, which should be a decreasing sigmoidal function. Perona & Malik suggested using $g(x) = e^{-|\nabla I|^2/2\mu}$, where μ is a positive, free parameter that controls the level of contrast of edges that can affect the smoothing process. Notice that $g(\| \nabla I \|)$ approaches 1 for $\| \nabla I \| \ll \mu$ and 0 for $\| \nabla I \| \gg \mu$. Edges are generally associated with large image gradients, and thus diffusion across edges is stopped while regions that are relatively flat undergo smoothing. A mathematical analysis shows that solutions to (4.1) can actually exhibit an inverse diffusion near edges, and can enhance or sharpen smooth edges that have gradients greater than μ [19].

Using anisotropic diffusion for surface processing with meshes was proposed in [13]. In that work, the authors move the surface according to a weighted sum of principle curvatures, where the weighting depends on the surface geometry. This is not a variational formulation, and is thereby not a generalization of (4.1). Because it performs a convex re-weighting of principle curvatures, it can reduce smoothing, but cannot exhibit sharpening of features. Furthermore, the level set implementation we are proposing allows us to produce results on significantly more complex surface shapes.

The generalization of anisotropic diffusion as in (4.1) to surfaces is achieved from variational principles by minimizing the penalty function

$$\mathscr{G} = \int_{\mathscr{S}} e^{-\frac{\kappa^2}{2\mu}} dx_{(i)}, \tag{4.2}$$

(a)

(b)

(c)

Figure 4.2: Processing results on the MRI head model: (a) original isosurface, (b)) isotropic diffusion of surface normals, and (c) anisotropic diffusion of surface normals. The small protrusion under the nose is a physical marker used for registration.

where κ^2 is computed from $N_{(i)}$ according to (3.4). The first variation with respect to the surface normals gives a vector-valued anisotropic diffusion on the level set surface—a straightforward generalization of (4.1). If we take the first variation with respect to ϕ, we obtain a fourth-order PDE, which is a modified version of ILMCF that preserves or enhances areas of high curvature, which we will call *creases*. Creases are the generalization of edges to surfaces. We will solve this variational problem by solving the second-order PDE on the normals using the framework introduced in Sec. 3. The strategy is the same as that in Fig. 3.3, where we replace $d\mathscr{G}/dN$ with the first variation of the penalty function from (4.2). This gives

$$T^N_{(ij)} \frac{d\mathscr{G}}{dN_{(j)}} = T^N_{(ij)} \left[g(\kappa) \left(N_{(j)k} - \frac{N_{(j)l}\phi_l\phi_k}{\phi_m\phi_m} \right) \right]_k \tag{4.3}$$

where κ is the total curvature as in (3.4), and the edge-stopping function becomes

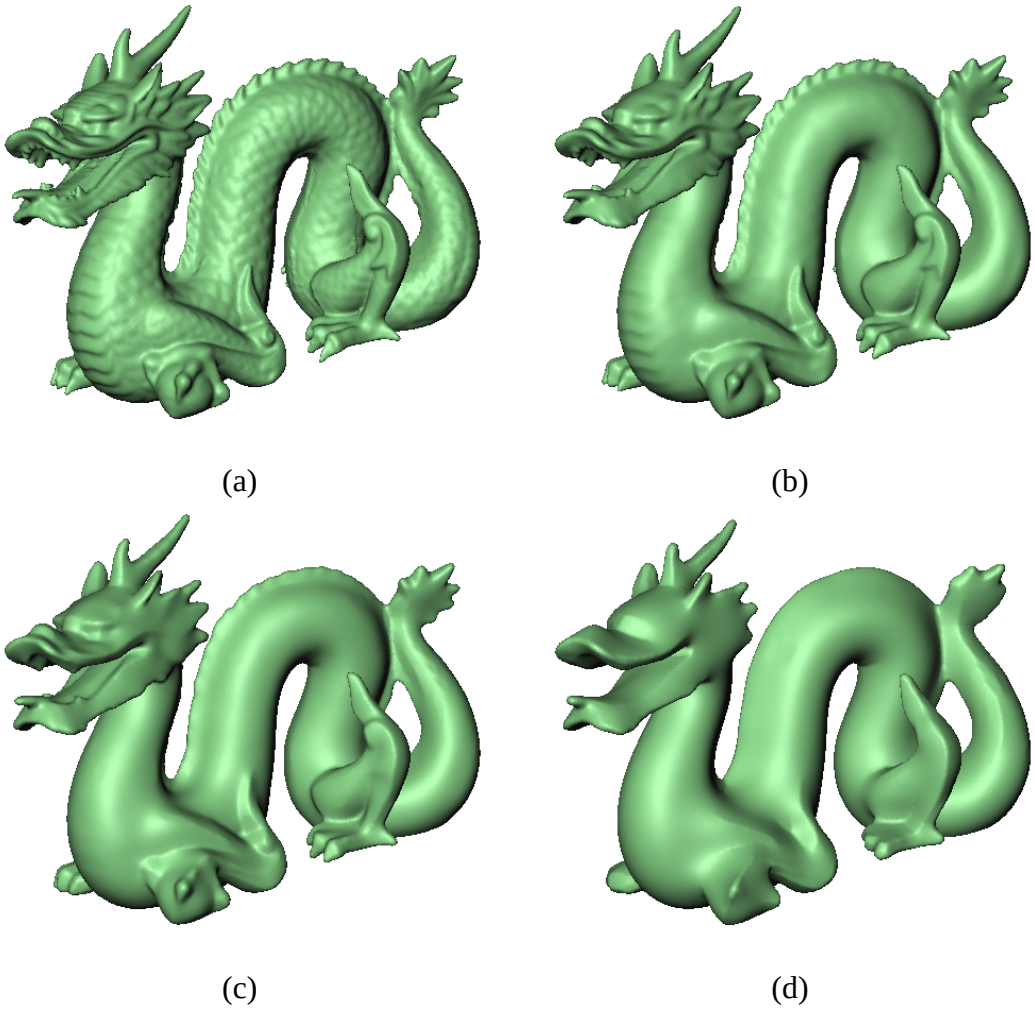
$$g(\kappa) = e^{-\frac{\kappa^2}{2\mu^2}}. \tag{4.4}$$

The rest of the process shown in Fig. 3.3 remains unchanged.

De-noising of measurement data is one of the most important applications of anisotropic diffusion. The differences between anisotropic diffusion and isotropic diffusion can clearly be observed in Fig. 4.2(c). Around the smooth areas of the original model such as the forehead and the cheeks, there is no noticeable difference in the results of the two processes. However, very significant differences exist around the lips and the eyes. The creases in these areas, which have been eliminated by isotropic diffusion, are preserved by the anisotropic process.

Running anisotropic diffusion for more time steps produces surfaces that have piecewise smooth normals (analogous to the behavior of the Perona and Malik equation for images), which corresponds to smooth, almost planar, surface patches bounded by sharp creases. Figure 4.3 shows the evolution of the dragon model with anisotropic diffusion. After a few iterations of the overall loop in Fig. 3.3 the smallest details, such as the scales of the dragon's skin, disappear, as in Fig. 4.3(b). After some more iterations in Fig. 4.3(c), the finest details of the face and the spikes on the ridge of the back have been suppressed. Going even further, observe that the surface is starting to become polyhedral in Fig. 4.3(d).

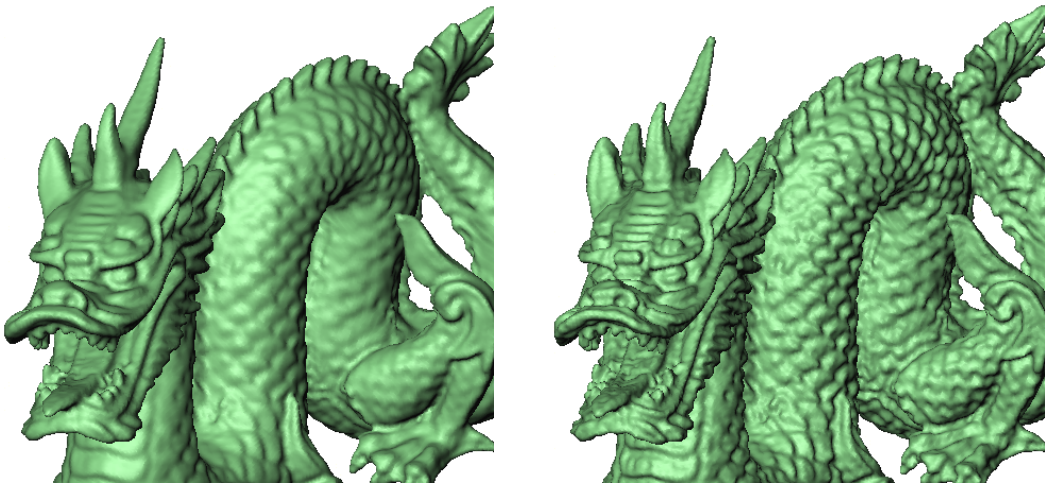
The non-linear progression of elimination of details from the smallest scale to the largest in Fig. 4.3 suggests a possibility for improved surface compression strategies, where details above a desired scale are kept effectively unchanged. This is in sharp contrast to isotropic diffusion and mean curvature flow, which distorts all features and generates sharp discontinuities as features disappear.



(a) (b)

(c) (d)

Figure 4.3: Various stages of anisotropic diffusion: (a) original model, (b) after 2 iterations, (c) after 6 iterations, and (d) after 26 iterations.



(a) (b)

Figure 4.4: Enhanced surface (a) after 1, and (b) 2 iterations of high-boost filtering.

These results support our proposition that processing the normals of a surface is the natural generalization of image processing. Processing normals on the surface is the same as processing a vector-valued image (such as a color image) with a distance metric that varies over the domain. This is in contrast to processing the points on a surface directly with lower-order flows, which offers no clear analogy to methods established in image processing.

4.2.1 High-boost filtering

The surface processing framework introduced in Sec. 3 is flexible and allows for the implementation of even more general image processing methods. We demonstrate this by describing how to generalize image enhancement by high-boost filtering to surfaces.

A high-boost filter has a frequency transform that amplifies high frequency components. In image processing this can be achieved by *unsharp masking* [26]. Let the low-pass filtered version of an image I be $\tilde{I}$. The high-frequency components are $\hat{I} = I - \tilde{I}$. The high-boost output is the sum of the input image and some fraction of its high-frequency components:

$$I_{\text{out}} = I + \alpha\hat{I} = (1+\alpha)I - \alpha\tilde{I}, \tag{4.5}$$

where α is a positive constant that controls the amount of high-boost filtering.

This same algorithm applies to surface normals by a simple modification to the flow chart in Fig. 3.3. Recall that the $d\mathscr{G}/dN$ loop produces $N_{(i)}^{n+1}$. Define a new set of normal vectors by

$$N'_{(i)} = \frac{(1+\alpha)N_{(i)}^{n} - \alpha N_{(i)}^{n+1}}{||(1+\alpha)N_{(j)}^{n} - \alpha N_{(j)}^{n+1}||}. \tag{4.6}$$

This new normal map is then input to the $d\mathscr{D}/d\phi$ catch-up loop. The effect of (4.6) is to extrapolate from the previous set of normals in the direction opposite to the set of normals obtained by isotropic diffusion. Recall that isotropic diffusion will smooth areas with high curvature and not significantly affect already smooth areas. Processing the loop with the modification of (4.6) will have the effect of increasing the curvature in areas of high curvature, while leaving smooth areas relatively unchanged. Thus we are able to obtain high quality surface enhancement on fairly complex surfaces of arbitrary topology, as in Figs. 4.4 and 4.5. The effects of high-boost filtering can be observed by comparing the original dragon model in Fig. 4.3 with the high-boost filtered model in Fig. 4.4. The scales on the skin and the ridge back are enhanced. Also, note that different amounts of enhancement can be achieved by controlling the number of iterations of the main loop. The degree of low-pass filtering used to obtain $N_{(i)}^{n+1}$ controls the size of the features that are enhanced. Figure 4.5 shows another example of high-boost filtering; notice the enhancement of features particularly on the wings.

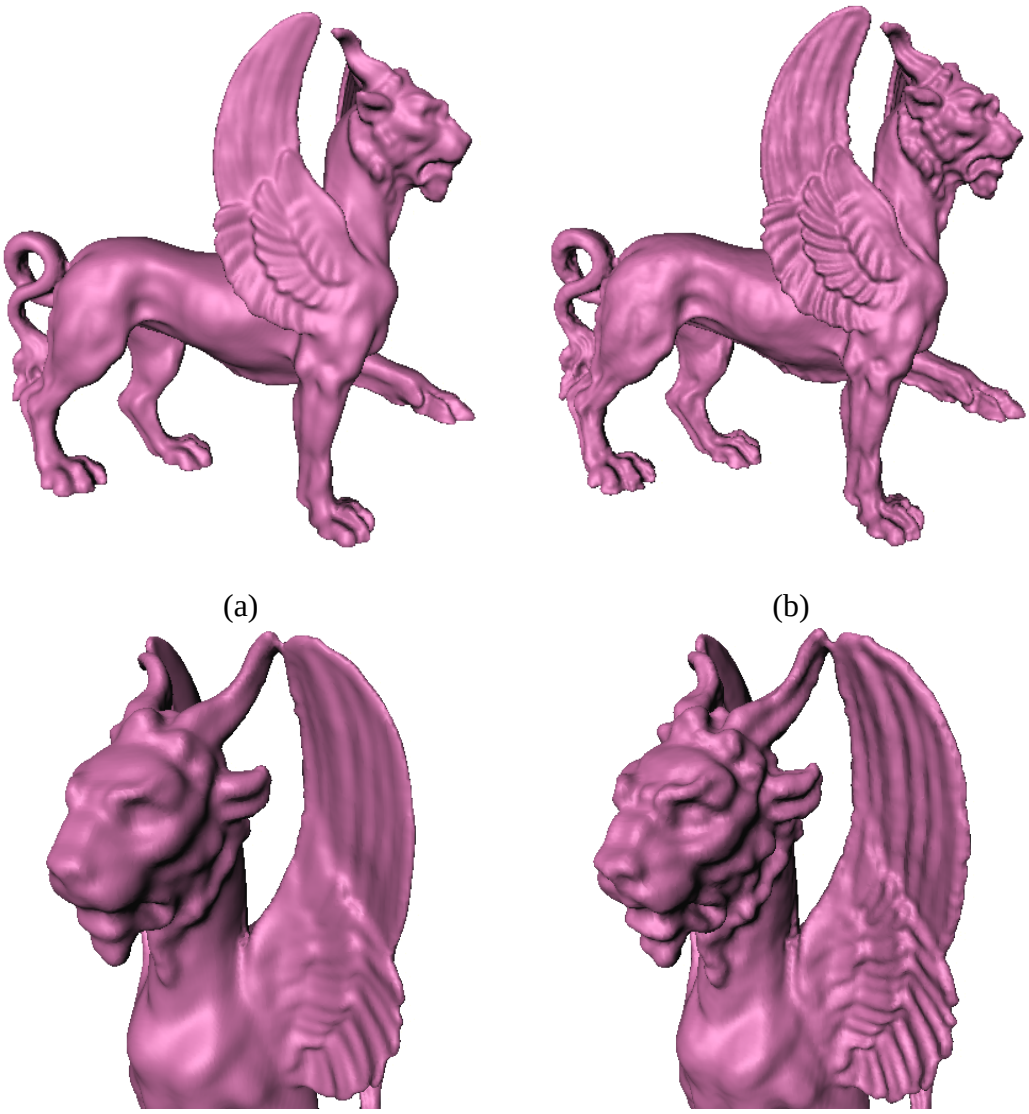


Figure 4.5: High-boost filtering: (a) original model, (b) after filtering, (c) close-up of original, and (d) filtered model.

Chapter 5

Conclusion

The *natural* generalization of image processing to surfaces is via the normals. Variational processes on the surface have corresponding variational formulations on the surface normals. The generalization of image-processing to surface normals, however, requires that we process the normals using a metric on the surface manifold, rather than a simple, flat metric, as we do with images. In this framework, the diffusion of the surface normals (and corresponding motions of the surface) is equivalent to particular fourth-order flow, the intrinsic Laplacian of mean curvature. By processing the normals separately, we can solve a pair of coupled second-order equations instead of a four-order equation. Typically, we allow one equation (the surface) to lag the other, but as the lag gets very small, it should not matter. We solve these equations using implicit surfaces, representing the implicit function on a discrete grid, modeling the deformation with the method of level sets. This level set implementation allows us to separate the shape of the model from the processing mechanism.

The method generalizes because we can do virtually anything we wish with the normal map. A generalization of anisotropic diffusion to a constrained, vector-valued function, defined on a manifold, gives us a smoothing process that preserves creases. If we want to enhance the surface, we can enhance the normals and allow the surface to catch up. Because of the implementation, the method applies equally well to surfaces that can be represented in a volume. Consequently, our results show a level of surface complexity that goes beyond that of previous methods.

Future work will study the usefulness of other interesting image processing techniques such as *total variation* [27] and local contrast enhancement. To date, we have dealt with post processing surfaces, but future work will combine this method with segmentation and re-

construction techniques. The current shortcoming of this method is the computation time, which is significant. However, the process lends itself to parallelism, and the advent of cheap, specialized, vector-processing hardware promises significantly faster implementations.

Bibliography

[1] W. Lorenson and H. Cline, "Marching cubes: A high resolution 3d surface reconstruction algorithm," in *Proceedings of SIGGRAPH'87*, pp. 163–169, 1987.

[2] I. Guskov, W. Sweldens, and P. Schroder, "Multiresolution signal processing for meshes," in *Proceedings of SIGGRAPH'99*, pp. 325–334, 1999.

[3] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE Trans. on PAMI*, vol. 17, no. 2, pp. 158–175, 1995.

[4] R. T. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int. J. Computer Vision*, vol. 29, no. 3, pp. 203–231, 1998.

[5] W. Welch and A. Witkin, "Free-form shape design using triangulated surfaces," in *Proceedings of SIGGRAPH'94*, pp. 247–256, 1994.

[6] G. Taubin, "A signal processing approach to fair surface design," in *Proceedings of SIGGRAPH'95*, pp. 351–358, 1995.

[7] M. Desbrun, M. Meyer, M. Schroder, and A. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proceedings of SIGGRAPH'99*, pp. 317–324, 1999.

[8] R. Schneider and L. Kobbelt, "Generating fair meshes with g^1 boundary conditions," in *Geometric Modeling and Processing Proceedings*, pp. 251–261, 2000.

[9] H. P. Moreton and C. H. Sequin, "Functional optimization for fair surface design," in *Proceedings of SIGGRAPH'92*, pp. 167–176, 1992.

[10] W. Welch and A. Witkin, "Variational surface modeling," in *Proceedings of SIGGRAPH'92*, pp. 157–166, July 1992.

[11] M. Halstead, M. Kass, and T. DeRose, "Efficient, fair interpolation using catmull-clark surface," in *Proceedings of SIGGRAPH'93*, pp. 35–44, 1993.

[12] R. Schneider and L. Kobbelt, "Geometric fairing of irregular meshes for free-form surface design." http://www-i8.informatik.rwth-aachen.de/geom_fair.pdf; To appear in Computer Aided Geometric Design, 2001.

[13] U. Clarenz, U. Diewald, and M. Rumpf, "Anisotropic geometric diffusion in surface processing," in *Proceedings of IEEE Visualization*, pp. 397–405, 2000.

[14] A. Polden, "Compact surfaces of least total curvature," tech. rep., University of Tubingen, Germany, 1997.

[15] G. Taubin, T. Zhang, and G. Golub, "Optimal surface smoothing as filter design," in *European Conf. on Computer Vision*, vol. 1, pp. 283–292, 1996.

[16] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel, "Interactive multi-resolution modeling on arbitrary meshes," in *Proceedings of SIGGRAPH'98*, pp. 105–114, 1998.

[17] D. L. Chopp and J. A. Sethian, "Motion by intrinsic laplacian of curvature," *Interfaces and Free Boundaries*, vol. 1, pp. 1–18, 1999.

[18] C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J. Verdera, "Filling-in by joint interpolation of vector fields and gray levels," *IEEE Trans. on Image Processing*, vol. 10, pp. 1200–1211, August 2001.

[19] G. Sapiro, *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press, 2001.

[20] S. Osher and J. A. Sethian, "Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulation," *J. Computational Physics*, vol. 79, pp. 12–49, 1988.

[21] J. A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Material Sciences*. Cambridge University Press, 1996.

[22] M. P. DoCarmo, *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.

[23] M. Bertalmio, L.-T. Cheng, S. Osher, and G. Sapiro, "Variational methods and partial differential equations on implicit surfaces," *J. Computational Physics*, vol. 174, pp. 759–780, 2001.

[24] N. Litke, A. Levin, and P. Schroder, "Fitting subdivision surfaces," in *Proceedings IEEE Visualization*, pp. 319–324, 2001.

[25] P. Perona and J. Malik, "Scale space and edge detection using anisotropic diffusion," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 629–639, July 1990.

[26] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Addison-Wesley Publishing Company, 1992.

[27] L. Rudin, S. Osher, and C. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, vol. 60, pp. 259–268, 1992.

[28] D. Adalsteinson and J. A. Sethian, "A fast level set method for propogating interfaces," *J. Computational Physics*, pp. 269–277, 1995.

[29] D. Peng, B. Merriman, S. Osher, H.-K. Zhao, and M. Kang, "A pde based fast local level set method," *J. Computational Physics*, vol. 155, pp. 410–438, 1999.

Appendix A

Numerical Implementation

By embedding surface models in volumes, we have converted equations that describe the movement of surface points to nonlinear, PDEs defined on a volume. The next step is to discretize these PDEs in space and time. In this paper, the embedding function ϕ is defined on the volume domain U and time. The PDEs are solved using a discrete sampling with forward differences along the time axis. There are two issues with discretization: (i) the accuracy and stability of the numerical solutions, (ii) the increase in computational complexity introduced by the dimensionality of the domain.

A.1 Notation

For brevity, we will discuss the numerical implementation in 2D— the extension to 3D is straightforward. The function $\phi : U \mapsto \mathbb{R}$ has a discrete sampling $\phi[p,q]$, where $[p,q]$ is a grid location and $\phi[p,q] = \phi(x_p, y_q)$. We will refer to a specific time instance of this function with superscripts, i.e. $\phi^n[p,q] = \phi(x_p, y_q, t_n)$. For a vector in 2-space v, we use $v_{(x)}$ and $v_{(y)}$ to refer to its components consistent with the notation of Sec. 3. In our calculations, we need three different approximations to first-order derivatives: forward, backward and central differences. We denote the type of discrete difference using superscripts on a difference operator, i.e., $\delta^{(+)}$ for forward differences, $\delta^{(-)}$ for backward differences, and δ for central differences. For instance, the differences in the x direction on a discrete grid with unit spacing are

$$\delta_x^{(+)}\phi[p,q] \quad \stackrel{\triangle}{=} \quad \phi[p+1,q] - \phi[p,q],$$

$$\begin{aligned} \delta_x^{(-)}\phi[p,q] &\triangleq \phi[p,q]-\phi[p-1,q], \text{ and} \\ \delta_x\phi[p,q] &\triangleq \frac{\phi[p+1,q]-\phi[p-1,q]}{2}, \end{aligned} \tag{A.1}$$

where the time superscript has been left off for conciseness. The application of these difference operators to vector-valued functions denotes componentwise differentiation.

A.2 Numerical solution to ILMCF

In describing the numerical solution to ILMCF, we will refer to the flow chart in Fig. 3.3 for one time step of the main loop. Hence, the first step in our numerical implementation is the calculation of the surface normal vectors from ϕ^n. Recall that the surface is a level set of ϕ^n as defined in (3.1). Hence, the surface normal vectors can be computed as the unit vector in the direction of the gradient of ϕ^n. The gradient of ϕ^n is computed with central differences as

$$\phi_i^n[p,q] \approx \begin{pmatrix} \delta_x\phi^n[p,q] \\ \delta_y\phi^n[p,q] \end{pmatrix}; \tag{A.2}$$

and the normal vectors are initialized as

$$N_{(i)}^{u=0}[p,q] = \phi_i^n[p,q] / \parallel \phi_k^n[p,q] \parallel . \tag{A.3}$$

Because ϕ^n is fixed and allowed to lag behind the evolution of $N_{(i)}$, the time steps in the evolution of $N_{(i)}$ are denoted with a different superscript, u. For this evolution, $\partial N_{(i)}/\partial t = -d\mathscr{G}/dN_{(i)}$ is implemented with smallest support area operators. For ILMCF $d\mathscr{G}/dN_{(i)}$ is given by (3.9) which we will rewrite here component by component. The Laplacian of a function can be applied in two steps, first the gradient and then the divergence. In 2D, the gradient of the normals produces a 2×2 matrix, and the divergence operator in (3.9) collapses this to a 2×1 vector. The diffusion of the normal vectors in the tangent plane of the level-sets of ϕ, requires us to compute the flux in the x and y directions, which we denote $M_{(ij)}^u = M_{(ix)}^u, M_{(iy)}^u$. The "columns" of the flux matrix are computed independently as

$$M_{(ix)}^u \approx \delta_x^{(+)}N_{(i)}^u - \left(N_{(i)j}^u \frac{\phi_j^n}{\parallel \phi_k^n \parallel} \right) \frac{\delta_x^{(+)}\phi^n}{\parallel \phi_k^n \parallel}, \tag{A.4}$$

$$M_{(iy)}^u \approx \delta_y^{(+)}N_{(i)}^u - \left(N_{(i)j}^u \cdot \frac{\phi_j^n}{\parallel \phi_k^n \parallel} \right) \frac{\delta_y^{(+)}\phi^n}{\parallel \phi_k^n \parallel} \tag{A.5}$$

where the time index n remains fixed as we increment u. Derivatives of the normal vectors are computed with forward differences; therefore they are staggered, located on a grid that

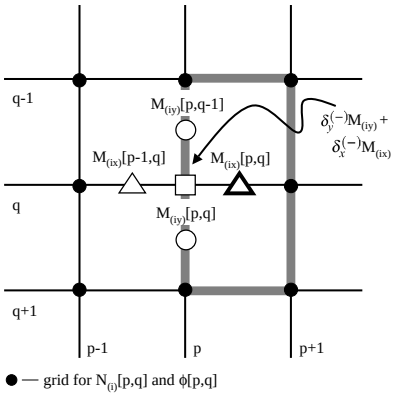


Figure A.1: Computation grid

is offset from the grid where ϕ and $N_{(i)}$ are defined, as shown Fig. A.1 for the 2D case. Furthermore, notice that since the offset is half pixel only in the direction of the differentiation, the locations of $\delta_x^{(+)} N_{(i)}$ and $\delta_y^{(+)} N_{(i)}$ are different, but are the same locations as $M^u_{(x)}$ and $M^u_{(y)}$ respectively. To evaluate (A.4) and (A.5), $\phi_j^n \; N^n_{(i)\,j}$ must computed at $[p+1/2, q]$ and $[p, q+1/2]$, respectively. These computations are also done with the smallest support area operators, using the symmetric 2×3 grid of samples around each staggered point, as shown in Fig. A.1 with the heavy rectangle. For instance,

$$\begin{aligned} \phi_j^n[p+\frac{1}{2}, q] &\approx \begin{pmatrix} \delta_x^{(+)} \phi[p,q] \\ \frac{1}{2}\left(\delta_y \phi[p,q] + \delta_y \phi[p+1,q]\right) \end{pmatrix}, \text{ and} \\ \phi_j^n[p, q+\frac{1}{2}] &\approx \begin{pmatrix} \frac{1}{2}\left(\delta_x \phi[p,q] + \delta_x \phi[p,q+1]\right) \\ \delta_y^{(+)} \phi[p,q] \end{pmatrix}. \end{aligned} \tag{A.6}$$

Backwards differences of the flux matrix are used to compute the divergence operation in (3.9)

$$\left[\frac{d\mathscr{G}}{dN_{(i)}}\right]^u \approx -\delta_x^{(-)} M^u_{(ix)} + \delta_y^{(-)} M^u_{(iy)} \tag{A.7}$$

Notice that backwards difference of $M^u_{(ix)}$, is defined at the original ϕ grid location $[p,q]$, and the same holds for $M^u_{(iy)}$. Thus all the components of $d\mathscr{G}/dN_{(i)}$ are located on the original grid for ϕ. Using the tangential projection operator in (3.9), the new set of normal vectors are computed as

$$N^{u+1}_{(i)} = N^u_{(i)} + T^N \left[\frac{d\mathscr{G}}{dN_{(i)}}\right]^u$$

$$= \quad N^u_{(i)} + \left[\frac{d\mathscr{G}}{dN_{(i)}}\right]^u - \left(\left[\frac{d\mathscr{G}}{dN_{(j)}}\right]^u N^u_{(j)}\right) N^u_{(i)}. \tag{A.8}$$

Starting with the initialization in (A.3) for $u = 0$, we iterate (A.8) for a fixed number of steps. In other words, we do not aim at minimizing the energy given in (3.8) in the $d\mathscr{G}/dN$ loop of Fig. 3.3; we only reduce it. The minimization of total mean curvature as a function of ϕ is achieved by iterating the main loop in (A.3). In Sec. 3.3, it was shown that to minimize total mean curvature, the $d\mathscr{G}/dN$ loop should be processed for only one time step before processing the $d\mathscr{D}/d\phi$ loop in every iteration of the main loop. However, the $d\mathscr{D}/d\phi$ loop is the most computationally expensive part of the algorithm and run-times can be reduced by running the main loop as few times as possible. To this effect, we found that the $d\mathscr{G}/dN$ loop can be processed for multiple time steps before moving onto the $d\mathscr{D}/d\phi$ loop. Moreover, the number of iterations of the main loop necessary to obtain the same result is reduced with this approach (25 iterations or less per main loop for the examples in this paper).

Once the evolution of N is concluded, ϕ is evolved to catch up with the new normal vectors according to (3.11). We denote the evolved normals by $N^{n+1}_{(i)}$. To solve (3.11) we must calculate H^{ϕ} and $H^{N^{n+1}}$. $H^{N^{n+1}}$ is the induced mean curvature of the normal map; in other words, it is the curvature of the hypothetical target surface that fits the normal map. Calculation of curvature from a field of normals is

$$H^{N^{n+1}} \approx \delta_x N^{n+1}_x + \delta_y N^{n+1}_y, \tag{A.9}$$

where we have used central differences on the components of the normal vectors. $H^{N^{n+1}}$ needs to be computed once at initialization as the normal vectors remain fixed during the catch up phase. Let v be the time step index in the $d\mathscr{D}/d\phi$ loop. H^{ϕ^v} is the mean curvature of the moving level set surface at time step v and is calculated from ϕ with the smallest area of support:

$$H^{\phi^v} \approx \delta^{(-)}_x \frac{\delta^{(+)}_x \phi^v[p,q]}{\| \phi^v_j[p+\frac{1}{2},q] \|} + \delta^{(-)}_y \frac{\delta^{(+)}_y \phi^v[p,q]}{\| \phi^v_j[p,q+\frac{1}{2}] \|} \tag{A.10}$$

where the off-grid gradients of ϕ in the denominator are calculated using the 2×3 staggered neighborhood as in (A.6).

The PDE in (3.11) is solved with a finite forward differences, but with the up-wind scheme for the gradient magnitude [20], to avoid overshooting and maintain stability. The up-wind method computes a one-sided derivative that looks in the up-wind direction of the moving wave front, and thereby avoids overshooting. Moreover, because we are interested in only a single level set of ϕ, solving (3.11) over all of U is not necessary. Because different level

sets evolve independently, we can compute the evolution of ϕ only in a narrow band around the level set of interest and re-initialize this band as necessary [28, 29]. See [21] for more details on numerical schemes and efficient solutions for level set methods.

Using the upwind scheme and narrow band methods, ϕ^{v+1} is computed from ϕ^{v} according to (3.11) using the curvatures computed in (A.9) and (A.10). This loop is iterated until the energy in (3.10) ceases to decrease; let v^{final} denote the final iteration of this loop. Then we set ϕ for the next iteration of the main loop (see Fig. 3.3) as $\phi^{n+1} = \phi^{v^{final}}$ and repeat the entire procedure. The number of iterations of the main loop is a free parameter that generally determines the extent of processing, i.e. the amount of smoothing for ILMCF.

Practical Animation of Liquids

Nick Foster*
PDI/DreamWorks

Ronald Fedkiw**
Stanford University

Abstract

We present a general method for modeling and animating liquids. The system is specifically designed for computer animation and handles viscous liquids as they move in a 3D environment and interact with graphics primitives such as parametric curves and moving polygons. We combine an appropriately modified semi-Lagrangian method with a new approach to calculating fluid flow around objects. This allows us to efficiently solve the equations of motion for a liquid while retaining enough detail to obtain realistic looking behavior. The object interaction mechanism is extended to provide control over the liquid's 3D motion. A high quality surface is obtained from the resulting velocity field using a novel adaptive technique for evolving an implicit surface.

Keywords: animation, computational fluid dynamics, implicit surface, level set, liquids, natural phenomena, Navier-Stokes, particles, semi-Lagrangian.

1. Introduction

The desire for improved physics-based animation tools has grown hand in hand with the advances made in computer animation on the whole. It is natural then, that established engineering techniques for simulating and modeling the real world have been modified and applied to computer graphics more frequently over the last few years. One group of methods that have resisted this transition are those used to model the behavior of liquids from the field of computational fluid dynamics (CFD). Not only are such techniques generally complex and computationally intensive, but they are also not readily adaptable to what could be considered the basic requirements of a computer animation system.

One of the key difficulties encountered when using these methods for animation directly characterizes the trade off between simulation and control. Physics-based animations usually rely on direct numerical simulation (DNS) to achieve realism. In engineering terms, this means that initial conditions and boundary conditions are specified and the process is left to run freely with only minor influence on the part of the animator. The majority of engineering techniques for liquid simulation assume this model.

From an animation viewpoint, we are interested in using numerical techniques to obtain behaviors that would be prohibitive to model by hand. At the same time we want control over the global, low frequency motion so we can match it to the behavior we are trying to create. This then becomes the goal when transitioning between engineering and computer animation; preserve as much of the realistic behavior as feasible while allowing for control over motion on both a local and global scale. This has to be achieved without compromising the overall requirement of a visually coherent and realistic look.

* nickf@pdi.com, ** fedkiw@cs.stanford.edu

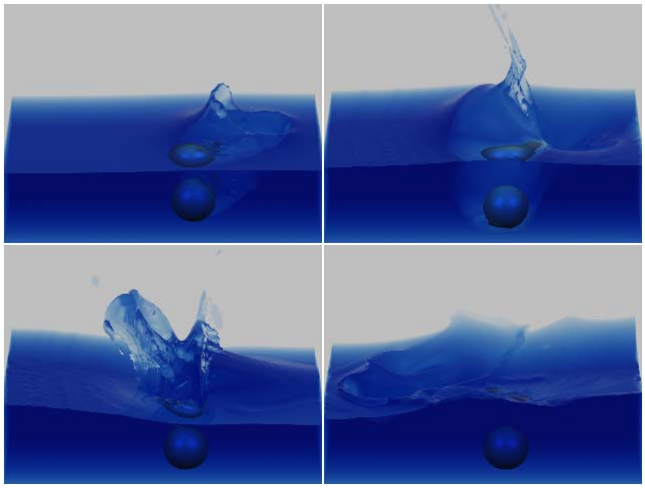


***Figure 1**: A ball splashes into a tank of water.*

This paper specifically addresses these issues for liquid animation. The method presented is for animating viscous liquids ranging from water to thick mud. These liquids can freely mix, move arbitrarily within a fixed three-dimensional grid and interact realistically with stationary or moving polygonal objects. This is achieved for animation by trading off engineering correctness for computational efficiency.

We start with the Navier-Stokes equations for incompressible flow and solve for liquid motion using an adaptation of a semi-Lagrangian method introduced recently to graphics for solving fluid flows [25]. These methods usually result in mass dissipation. While not an issue for gas or smoke, this is visually unacceptable for modeling liquids. We correct for this by tracking the motion of the liquid surface using a novel hybrid combination of inertialess particles and an implicit surface called a level set. The level set prevents mass dissipation while the particles allow the liquid to still splash freely. A useful consequence is that this combined surface can be rendered in a highly believable way.

The next innovation involves taking account of the effects of moving polygonal objects within the liquid. We develop a new technique that, while not accurate in an engineering sense, satisfies the physics of object/liquid interactions and looks visually realistic. This method is efficient and robust, and we show that it can be adapted to provide low frequency directional control over the liquid volume. This allows us to efficiently calculate liquid behavior that would be impossible to get by hand,

while at the same time allowing us to "dial-in" specific motion components.

When the techniques described above are applied together, the result is a comprehensive system for modeling and animating liquids for computer graphics. The main contributions of the system are a numerical method that takes the minimal computational effort required for visual realism combined with tailor-made methods for handling moving objects and for maintaining a smooth, temporally coherent liquid surface.

2. Previous Work

The behavior of a volume of liquid can be described by a set of equations that were jointly developed by Navier and Stokes in the early eighteen hundreds (see next section). The last fifty years has seen an enormous amount of research by the CFD community into solving these equations for a variety of engineering applications. We direct the interested reader to Abbot and Basco [1] which covers some of the important principles without being too mathematically dense.

Early graphics work concentrated on modeling just the surface of a body of water as a parametric function that could be animated over time to simulate wave transport [12, 22, 23]. Kass and Miller [17] approximated the 2D shallow water equations to get a dynamic height field surface that interacted with a static ground "object". Chen and Lobo [4] extended the height field approach by using the pressure arising from a 2D solution of the Navier-Stokes equations to modulate surface elevation. O'Brien and Hodgins [20] simulated splashing liquids by combining a particle system and height field, while Miller and Pearce [19] used viscous springs between particles to achieve dynamic flow in 3D. Terzopoulos, Platt and Fleischer [27] simulated melting deformable solids using a molecular dynamics approach to simulate the particles in the liquid phase.

Surface or particle based methods are relatively fast, especially in the case of large bodies of water, but they don't address the full range of motion exhibited by liquids. Specifically, they don't take advantage of the realism inherent in a full solution to the Navier-Stokes equations. They are also not easily adapted to include interaction with moving objects. Foster and Metaxas [11] modified an original method by Harlow and Welch [15] (later improved by others, see e.g. [5]) to solve the full equations in 3D with arbitrary static objects and extended it to include simple control mechanisms [9]. Foster and Metaxas also applied a similar technique to model hot gases [10]. Stam [25] replaced their finite difference scheme with a semi-Lagrangian method to achieve significant performance improvements at the cost of increased rotational damping. Yngve et al. used a finite difference scheme to solve the compressible Navier-Stokes equations to model shock wave and convection effects generated by an explosion [28].

3. Method Outline

The Navier-Stokes equations for describing the motion of a liquid consist of two parts. The first, enforces incompressibility by saying that mass should always be conserved, i.e.

$$\nabla \cdot \mathbf{u} = 0 , \qquad (3.1)$$

where $\mathbf{u}$ is the liquid velocity field, and

$$\nabla = \left(\partial / \partial \mathrm{x}, \partial / \partial \mathrm{y}, \partial / \partial \mathrm{z} \right)$$

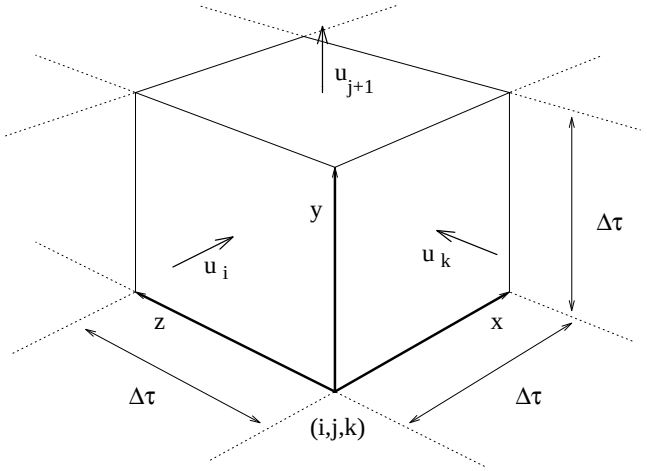


Figure 2: *A single grid cell with three of its six face velocities shown.*

is the gradient operator. The second equation couples the velocity and pressure fields and relates them through the conservation of momentum, i.e.

$$\mathbf{u}_t = \nu \nabla \cdot \left(\nabla \mathbf{u} \right) - \left(\mathbf{u} \cdot \nabla \right) \mathbf{u} - \frac{1}{\rho} \nabla \mathrm{p} + \mathbf{g} . \qquad (3.2)$$

This equation models the changes in the velocity field over time due to the effects of viscosity (ν), convection, density (ρ), pressure (p), and gravity (**g**). By solving (3.1) and (3.2) over time, we can model the behavior of a volume of liquid. The new algorithm we are proposing to do this consists of six straightforward steps.

I. Model the static environment as a voxel grid.

II. Model the liquid volume using a combination of particles and an implicit surface.

Then, for each simulation time step

III. Update the velocity field by solving (3.2) using finite differences combined with a semi-Lagrangian method.

IV. Apply velocity constraints due to moving objects.

V. Enforce incompressibility by solving a linear system built from (3.1).

VI. Update the position of the liquid volume (particles and implicit surface) using the new velocity field.

These steps are described in detail in the following sections. Steps IV and V are presented in reverse order for clarity.

4. Static Environment

Equations (3.1) and (3.2) model a liquid as two coupled dynamic fields, velocity and pressure. The motion of the liquid we are modeling will be determined by evolving these fields over time. We start by representing the environment that we want the liquid to move in as a rectangular grid of voxels with side length Δτ. The grid does not have to be rectangular, but the overhead of unused (non-liquid containing) cells will be low and so it is convenient. Each cell has a pressure variable at its center and shares a velocity variable with each of its adjacent neighbors (see figure 2). This velocity is defined at the center of the face shared

by the two neighboring cells and represents the magnitude of the flow normal to that face. This is the classic "staggered" MAC grid [15]. Each cell is then either tagged as being empty (available to be filled with liquid) or filled completely with an impermeable static object. Despite the crude voxelized approximation of both objects and the liquid volume itself, we'll show that we can still obtain and track a smooth, temporally coherent liquid surface.

5. Liquid Representation

The actual distribution of liquid in the environment is represented using an implicit surface. The implicit function is derived from a combination of inertialess particles and a dynamic isocontour. The isocontour provides a smooth surface to regions of liquid that are well resolved compared to our grid, whereas the particles provide detail where the surface starts to splash.

5.1 Particles

Particles are placed (or introduced via a source) into the grid according to some initial liquid distribution. Their positions then evolve over time by simple convection. Particle velocity is computed directly from the velocity grid using tri-linear interpolation and each particle is moved according to the inertialess equation $d\mathbf{x}_p/dt = \mathbf{v}_x$, where $\mathbf{v}_x$ is the fluid velocity at $\mathbf{x}_p$. Particles have a low computational overhead and smoothly integrate the changing liquid velocity field over time. The obvious drawback to using them, however, is that there is no straightforward way to extract a smooth polygonal (or parametric) description of the actual liquid surface. This surface is preferred because we want to render the liquid realistically using traditional computer graphics techniques. It is possible to identify it by connecting all the particles together into triangles, although deducing both the connectivity and set of surface triangles is difficult. In addition, since the particles do not generally form a smooth surface, the resulting polygonal mesh suffers from temporal aliasing as triangles "pop" in or out.

5.2 Isocontour

An alternative technique for representing the liquid surface is to generate it from an isocontour of an implicit function. The function is defined on a high resolution Eulerian sub-grid that sits inside the Navier-Stokes grid. Let each particle represent the center of an implicitly defined volumetric object (see Bloomenthal et al. [3] for a survey of implicit surfaces). Specifically, an implicit function centered at the particle location $\mathbf{x}_p$ with radius r is given by

$$\phi_p(\mathbf{x}) = \sqrt{(x_i - x_{pi})^2 + (x_j - x_{pj})^2 + (x_k - x_{pk})^2} - r .$$

The surface of that particle is defined as the spherical shell around $\mathbf{x}_p$ where $\phi_p(\mathbf{x})=0$. An implicit function, $\phi(\mathbf{x})$, is then defined over all the particles by taking the value of $\phi_p(\mathbf{x})$ from the particle closest to $\mathbf{x}$. If we sample $\phi(\mathbf{x})$ at each sub-grid point we can use a marching cubes algorithm [18] to tessellate the $\phi(\mathbf{x})=0$ isocontour with polygons. More sophisticated blend functions could be used to create an implicit function, however, we are going to temporally and spatially smooth $\phi(\mathbf{x})$ so it isn't necessary. We refer those interested in wrapping implicit surfaces around particles to the work of Desbrun and Cani-Gascuel [7].

The first step towards smoothing the surface is to normalize ϕ so that $|\phi(\mathbf{x})|$ equals the distance from $\mathbf{x}$ to the closest point on the zero isocontour. The sign of ϕ is set negative inside the liquid and positive outside. This signed distance function can be created quickly using the Fast Marching Method [24] starting from the initial guess of $\phi(\mathbf{x})$ defined by the particles as outlined above.

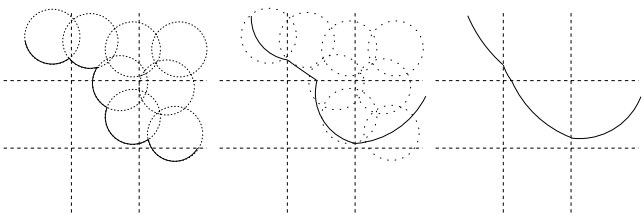


Figure 3: *The isocontour due to the implicit function around the particles, interpolated ϕ values, and smoothed ϕ values, respectively.*

In order to smooth out ϕ to reduce unnatural "folds" or "corners" in the surface (see figure 3), a smoothing equation of the form

$$\phi_\eta = -S\left(\phi^{\eta=0}\right)\left(\left|\nabla\phi\right| - 1\right), \qquad (5.1)$$

is used to modify values of ϕ close to the $\phi(\mathbf{x})=0$ isocontour. $S(\phi)$ is a smoothed sign function given by

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + \Delta\tau^2}} .$$

If applied for a few relaxation steps in fictitious time η (everything else remains constant), (5.1) smooths out the $\phi(\mathbf{x})=0$ isocontour while maintaining overall shape. Once smoothed, the isocontour can be ray traced directly using a root finding algorithm to identify the zero values of ϕ. A fast root finder can be built easily because at any sub-grid point the value of ϕ explicitly gives the minimum step to take to get closer to the surface. Note that the surface normal is given by $\mathbf{n} = \nabla\phi / \left|\nabla\phi\right|$.

By creating a smooth isocontour for each frame of animation, we get an improved surface representation compared to using particles alone. There are still drawbacks however. A high density of particles is required at the $\phi(\mathbf{x})=0$ isocontour before the surface looks believably flat. Particles are also required throughout the entire liquid volume even when it's clear that they make no contribution to the visible surface. The solution is to create ϕ once using the particles, and then track how it moves using the same velocity field that we're using to move the particles. This leads to a temporally smoothed dynamic isosurface known in the CFD literature as a level set.

5.3 Dynamic Level Set

An obvious way to track the evolution of the surface of a volume of liquid would be to attach particles directly to the surface in its initial position and then just move them around in the velocity field. This would require adding extra particles when the surface becomes too sparsely resolved, and removing them as the surface folds, or "splashes" back over itself. An alternative method which is intuitively similar, but that doesn't use particles, was developed by Osher and Sethian [21] and is called the level set method.

We want to evolve ϕ directly over time using the liquid velocity field $\mathbf{u}$. We have a smooth surface but need to conform, visually at least, to the physics of liquids. It has been shown [21] that the

equation to update ϕ under these circumstances has the following structure,

$$\phi_t + \mathbf{u} \cdot \nabla \phi = 0 \,. \qquad (5.2)$$

Using (5.2), the surface position is evolved over time by tracking $\phi(\mathbf{x})=0$. The $(\mathbf{u} \cdot \nabla \phi)$ term is a convection term similar to the $(\mathbf{u} \cdot \nabla)\mathbf{u}$ term in (3.2) implying that we could use a semi-Lagrangian method to solve this equation. However, since this equation represents the mass evolution of our liquid, the semi-Lagrangian method tends to be too inaccurate. Instead we use a higher order upwind differencing procedure [1] on the $(\mathbf{u} \cdot \nabla \phi)$ term. Fedkiw et al. [8] used this methodology to track a fluid surface and give explicit details on solving (5.2). This method can suffer from severe volume loss especially on the relatively coarse grids commonly used in computer graphics. This is clearly visible when regions of liquid break away during splashing and then disappear because they are too small to be resolved by the level set. We require visual coherency for this to be a useful graphics technique and so the level set method needs to be modified to preserve volume.

5.4 Hybrid Surface Model

Particle evolution is a fully Lagrangian approach to the mass motion while level set evolution is a fully Eulerian approach. Since they tend to have complementary strengths and weakness, a combined approach gives superior results under a wider variety of situations. Level set evolution suffers from volume loss near detailed features while particle evolution suffers from visual artifacts in the surface when the number of particles is small. Conversely, the level set is always smooth, and particles retain detail regardless of flow complexity. Therefore we suggest a novel combination of the two approaches.

At each time step we evolve the particles and the level set ϕ forward in time. Next, we use the updated value of the level set function to decide how to treat each particle. If a particle is more than a few grid cells away from, and inside the surface, as indicated by the locally interpolated value of ϕ, then that particle is deleted. This increases efficiency since particles are only needed locally near the surface of the liquid as opposed to throughout the entire liquid volume. In addition, for cells close to $\phi(\mathbf{x})=0$ that are sparsely populated, extra particles can be introduced "within" the isocontour. Thus, for a bounded number of particles, we get improved surface resolution.

Next, for each particle near the surface, the locally interpolated curvature of the interface, calculated as

$$k = \nabla \cdot \left(\nabla\phi \Big/ \left|\nabla\phi\right| \right),$$

is used to indicate whether or not the surface is smooth. Smooth regions have low curvature and the particles are ignored allowing the level set function to give a very smooth representation of the liquid surface. On the other hand, regions of high curvature indicate splashing. In these regions, the particles are a better indicator of the rough surface topology. Particles in these regions are allowed to modify the local values of ϕ. At grid points where the implicit basis function for the particle would give a smaller value of ϕ (i.e. a particle is "poking" out of the zero level set), this smaller value is used to replace the value obtained from the time evolution of ϕ.

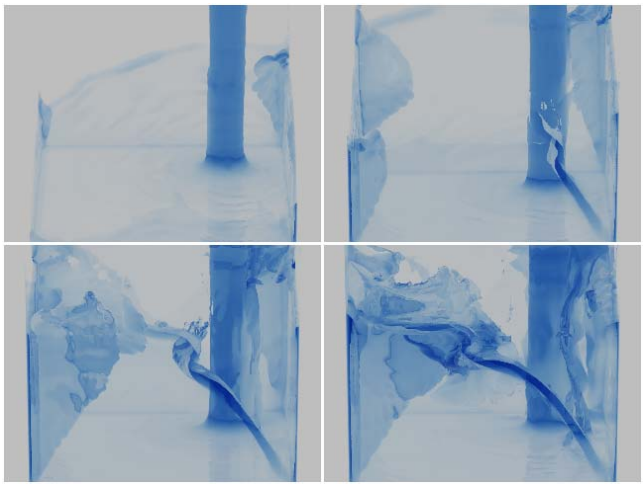


Figure 4: *Water pours into a container causing a complex surface to develop.*

Even with the tight coupling between the particles and the level set, some particles will escape the inside of the liquid layer since the grid is too coarse to represent them individually. These particles can be rendered directly as small liquid drops. In addition, these stray particles could be used as control particles to indicate the presence of fine spray or mist.

6. Updating the Velocity Field

We have a representation of the graphics environment and a way of tracking the surface of a volume of liquid. We can now apply (3.2) to the existing velocity field to advance it through an Euler-integration time step Δt. The equation is solved in three stages. First we compute Δt using the CFL condition (see Appendix A). Next, we update the convective component, i.e. $(\mathbf{u} \cdot \nabla)\mathbf{u}$, using a first order semi-Lagrangian method, as per Courant et al. [6] and by Stam [25]. We use the same formulation as Stam and refer readers to his description. Standard central differencing is then used on the viscous terms of (3.2) as described by Foster and Metaxas [11]. The results from this and the preceding calculation are added together to get an updated (though not mass conserving) velocity field for time $t+\Delta t$.

Semi-Lagrangian methods allow us to take large time steps without regard for the sometimes overly restrictive CFL condition [26]. Unfortunately, these large time steps come at the cost of added dissipation. This is visually acceptable for gases such as smoke where it appears realistic. For liquids however, mass dissipation ruins the visual effect. Therefore, even though we use a semi-Lagrangian method to update (3.2), the time step for evolving the particles and the level set still needs to be limited according to a plausible CFL condition. Updating the surface position isn't particularly expensive computationally, and so we alternate between a large time step for updating the Navier-Stokes equations and a series of small time steps (that add up to the large time step) for the particles and the level set. Our experience suggests that the velocity field time step can only be a few (around five) times bigger than that dictated by the usual CFL criterion. However, even this gives tremendous computational savings, since enforcing incompressibility (step V, discussed in section 8) is the most expensive part of the algorithm.

We caution the reader that using a particle only evolution with the semi-Lagrangian method introduces noise into the surface, and that using a level set only evolution with the semi-Lagrangian method gives noticeable volume loss. The key to making the semi-Lagrangian method work for liquids is the mixed Eulerian-Lagrangian formulation that uses *both* particles and level sets to evolve the surface position over time.

7. Boundary Conditions

When solving (3.2) within the grid, we need to specify pressure and velocity values for certain cells. We want stationary object cells to resist liquid motion and cells that represent the boundary between air and liquid to behave appropriately.

7.1.1 Non-liquid Cells

Cells in the grid that don't contain particles and aren't contained within the isosurface are either considered empty (open air) or are part of an object. If a cell is empty, its pressure is set to atmospheric pressure, and the velocity on each of its faces shared with another empty cell is set to zero. This assumes that air dynamics has a negligible effect. An object cell, on the other hand, can have velocities and pressures set using many different combinations to approximate liquid flowing into or out of the environment, or to approximate different object material properties. Foster and Metaxas [10] summarize and discuss methods to do this.

7.1.2 Liquid Surface

Other grid cells that require special attention are those that contain part of the $\phi(\mathbf{x})=0$ isocontour. Such cells represent what we know about the location of the liquid surface within the grid. The movement of the isocontour will determine how the surface evolves, but we need to set velocities on faces between empty and liquid cells so that normal and tangential stresses are zero. Intuitively, we need to make sure that the "air" doesn't mix with or inhibit the motion of the liquid, while allowing it to flow freely into empty cells. This is done by explicitly enforcing incompressibility within each cell that contains part of the liquid surface. Velocities adjacent to a liquid filled cell are left alone, whereas the others are set directly so (3.1) is satisfied for that cell. The pressure in a surface cell is set to atmospheric pressure.

8. Conservation of Mass

The velocity field generated after evolving the Navier-Stokes equations (steps III and IV) has rotation and convection components that are governed by (3.2) (excluding the pressure term). However, (3.1), conservation of mass, is only satisfied in surface cells where we have explicitly enforced it. The best we can do to preserve mass within our grid is to ensure that the incompressibility condition is satisfied for every grid cell (at least to some tolerance). Foster and Metaxas [11] achieved this using a technique called Successive Over Relaxation.

A more efficient method for enforcing incompressibility comes from solving the linear system of equations given by using the Laplacian operator to couple local pressure changes to the divergence in each cell. Specifically, this gives

$$\nabla^2 p = \rho \nabla \cdot \mathbf{u} / \Delta t \,, \qquad (8.1)$$

where $\nabla^2 p$ is the spatial variation (Laplacian) of the pressure and $\mathbf{u}$ is the velocity field obtained after solving (3.2). Applied at the center of a cell, (8.1) can be discretized as

$$\sum_{n=\{ijk\}} (p_{n+1} + p_{n-1}) - 6p = \rho \frac{\Delta \tau}{\Delta t} \sum_{n=\{ijk\}} (u_{n+1} - u_n), \qquad (8.2)$$

where $p_{n \pm 1}$ is the pressure from the cell ahead (+) or behind (-) in the n direction, and the u values are taken directly from the grid cell faces. Using (8.2), we form a linear system AP = b where P is the vector of unknown pressures needed to make the velocity field divergence free, b is the RHS of (8.2), and A has a regular but sparse structure. The diagonal coefficients of A, a_{ii}, are equal to the negative number of liquid cells adjacent to $cell_i$ (e.g., -6 for a fully "submerged" cell) while the off diagonal elements are simply $a_{ij}=a_{ji}=1$ for all liquid $cells_j$ adjacent to $cell_i$.

Conveniently, the system described above is symmetric and positive definite (as long as there is at least one surface cell as part of each volume). Static object and empty cells don't disrupt this structure. In that case pressure and velocity terms can disappear from both sides of (8.2), but the system remains symmetric. Because of this, it can be solved quickly and efficiently using a Preconditioned Conjugate Gradient (PCG) method. Further efficiency gains can be made by using an Incomplete Choleski preconditioner to accelerate convergence. There is a wealth of literature available regarding PCG techniques and we recommend any of the standard implementations, see Barret et al. [2] for some basic templates. Once the new pressures have been determined, the velocities in each cell are updated according to

$$u_{\{ijk\}}^{t+\Delta t} = u_{\{ijk\}} - \frac{\Delta t}{\rho \Delta \tau} (p_n - p_{n-1})$$

The resulting velocity field conserves mass (is divergence free) and satisfies the Navier-Stokes equations.

9. Moving Objects

Previous techniques proposed for liquid animation could deal with static objects that have roughly the same resolution as the grid, but they have difficulty dealing with moving objects. Unfortunately, the CFD literature has little to offer to help resolve the effects of moving objects on a liquid in terms of animation. There are sophisticated methods available for handling such interactions, but they typically require highly resolved computational grids or a grid mechanism that can adapt itself to the moving object surface. Neither approach is particularly well suited to animation because of the additional time complexity involved. Therefore, we propose the following method for handling interactions between moving objects and the liquid.

Consider an object (or part of an object) moving within a cell that contains liquid. There are two basic conditions that we want to enforce with respect to the computational grid, and an additional condition with respect to the surface tracking method. These are

1. Liquid should not flow into the object. At any point of contact, the relative velocity between the liquid and object along the object's surface normal should be greater than or equal to zero.
2. Tangential to the surface, the liquid should flow freely without interference.
3. Neither the particles nor the level set surface should pass through any part of the surface of the object.

The last of these is relatively straightforward. We know where the polygons that comprise the object surface are and in what direction they are moving. We simply move the particles so that they are always outside the surface of the object. As long as we accurately take account of the velocity field within the grid then the isocontour will remain in the correct position relative to the object.

To prevent liquid from flowing into the object we directly set the component of liquid velocity normal to the object. We know the object surface normal, $\mathbf{n}_s$, and can calculate the liquid velocity relative to that surface, $\mathbf{v}_r$, in a given cell. If $\mathbf{v}_r.\mathbf{n}_s < 0$ then liquid is flowing through the surface. In such cases we manipulate $\mathbf{u}$ in the cell so that $\mathbf{v}_r.\mathbf{n}_s = 0$ leaving the tangential ("slip") part of the velocity unchanged.

These velocities need to be applied without introducing visual artifacts into the flow. The following method solves for both normal and tangential velocity components. It's relatively intuitive, and it seems to work well in practice. The steps are

1. As a boundary condition, any cell within a solid object has its velocities set to that of the moving object.
2. The velocity field is updated using (3.2). No special consideration is given to cells containing an object, i.e. they are all allowed to change freely as if they contain liquid.
3. Each cell that intersects an object surface gets the component of the object velocity along its normal set explicitly as outlined above.
4. Cells internal to the object have their velocities set back to the object velocity.
5. During the mass conservation step (section 8) the velocity for any grid cell that intersects the object is held fixed.

The result of this approach is that liquid is both pushed along by an object while being allowed to flow freely around it, causing realistic-looking behavior in the mean time. Obviously it's only possible to accurately account for one polygon face per grid cell. Objects that are more detailed with respect to the grid can still be handled by determining an average object surface normal and velocity for each intersecting cell, but grid resolution remains the limiting factor.

10. Control

Animation is all about control. Having things behave according to some arbitrary aesthetic is the goal of most production software. The difficulty is in providing this level of control over a physics-based animation while still maintaining a realistic behavioral component. The nature of the governing equations of motion of liquids means that they will always swirl, mix, and splash unless the applied forces are identical everywhere. This necessarily limits the level of control that we can have over the final motion and comes with the territory of non-linear simulation.

Gates [13] has shown that mass conserving flow fields can be blended with calculated fields to get good non-dynamic results. The Navier-Stokes equations allow for the body force term, $\mathbf{g}$, to be manipulated directly [9] much like a traditional particle system. Forces aren't always a very intuitive way of getting motion that we want however. The moving object mechanism on the other hand, is well suited to this. Instead of moving polygons, we can explicitly set velocities anywhere in the grid by introducing "fake" surfaces (a single point even) that have normals and velocities pointing in the direction that we want the liquid to go. Setting the normal *and* tangential velocities in individual cells is also possible if it is done before the mass conservation calculation. This allows the solver to smooth out any lack of physical correctness in applied velocities before passing them into (3.2).

As a brief example, consider a set of 3D parametric space curves that define the desired path for the liquid to follow. We instance a set of points along each curve giving each point a parametric coordinate φ_p. A point's spatial position is then given simply by the curve definition, i.e. $\mathbf{x}_p=F(\varphi_p)$. The velocity of the point can then be described as

$$\mathbf{v}_p = C(t)\left(dF(\varphi_p)/d\varphi_p\right)$$

where $C(t)$ is a monotonic key framed scaling function. $C(t)$ is also used to update φ_p over time according to $d\varphi_p/dt = C(t)$. The "fake" surface normal of the point is then simply $\mathbf{n}_p = \mathbf{v}_p/|\mathbf{v}_p|$. By manipulating $\mathbf{x}_p$, $\mathbf{v}_p$, and $\mathbf{n}_p$ over time, we can "trap" small pockets of liquid and control them directly. The governing equations then make sure that neighboring liquid attempts to follow along.

This basic approach can be adapted to surfaces or even volumetric functions as long as they vary smoothly. While still not giving perfect direct control over the liquid motion, when combined with force fields it is good enough to make it a useful animation tool.

11. Results

The animation system described in the preceding sections was used to generate all of the examples in this paper. The basic Navier-Stokes solver and implicit surface are demonstrated by the container-filling example in figure 4. The combination of particles and level set make sure that the resulting surface stays smooth and behaves in a physically believable way. The splashing object examples in figures 1, 5 and 6 show close interaction between the liquid and moving objects. They also show how the hybrid surface can handle extreme splashing without either the particles or level set being apparent. The particles play a large role in both cases by allowing the liquid to "splash" at a higher resolution than would be possible with the level set alone. All of these images were rendered using a ray-tracing algorithm that marches through the implicit surface grid as outlined in section 5.

The final example, figure 7, makes use of just a spherical implicit function around each particle. It shows the interaction between a thick (high viscosity) liquid and a hand animated character. The character surface is sampled at each grid cell and the mechanisms described in section 9 take account of all the motion in the scene. This includes the character filling his mouth with mud. The mud is later ejected using a 3D space curve as a controller as outlined in section 10. The captions to each figure give the static grid size used during calculation along with computation times per frame (for motion, not rendering) on a PentiumII 500MHz.

12. Conclusion

We have presented a method for modeling and animating liquids that is a pragmatic compromise between the numerical care that needs to be taken when simulating non-linear physics and the interaction and control animators require. Where appropriate, we have drawn on techniques from computational fluid dynamics and combined them with recent computer graphics advances as well

as new methods for free surface evolution and interaction between moving objects and the liquid volume. The result is a technique that is very general, efficient, and offers flexible control mechanisms for specifying how the liquid should behave.

13. Acknowledgements

The work of the second author was supported in part by ONR N00014-97-1-0027.

A. Courant-Friedrichs-Levy (CFL) Condition

The CFL condition is a restriction on the size of the time step, Δt, that can be used together with a time-marching numerical simulation. It says that Δt must be less than the minimum time over which "something significant" can occur in the system for the simulation to remain numerically stable. The CFL condition depends both on the physical system being modeled as well as the specifics of the discretization method employed. In the context of the system described in this paper a good CFL condition is that a discrete element of liquid cannot "jump over" a cell in the computational grid, i.e. $\Delta t < \Delta\tau / |\mathbf{u}|$.

Note that the viscosity related terms also impose a CFL type restriction. This can be avoided by locally adjusting the magnitude of the viscosity in cells where the viscous terms would dictate the necessity for a smaller time step.

References

[1] Abbot, M. and Basco, D., "Computational Fluid Dynamics – An Introduction for Engineers", Longman, 1989.

[2] Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C. and van der Vorst, H., "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", Society for Industrial and Applied Mathematics, 1993.

[3] Bloomenthal, J., Bajaj, C., Blinn, J., Cani-Gascual, M.-P., Rockwood, A., Wyvill, B. and Wyvill, G., "Introduction to Implicit Surfaces", Morgan Kaufmann Publishers Inc., San Francisco, 1997.

[4] Chen, J. and Lobo, N., "Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using the Navier-Stokes Equations", Graphical Models and Image Processing 57, 107-116 (1994).

[5] Chen, S., Johnson, D., Raad, P. and Fadda, D., "The Surface Marker and Micro Cell Method", Int. J. Numer. Methods in Fluids 25, 749-778 (1997).

[6] Courant, R., Issacson, E. and Rees, M., "On the Solution of Nonlinear Hyperbolic Differential Equations by Finite Differences", Comm. Pure and Applied Math 5, 243-255 (1952).

[7] Desbrun, M. and Cani-Gascuel, M.P., "Active Implicit Surface for Animation", Graphics Interface 98, 143-150 (1998).

[8] Fedkiw, R., Aslam, T., Merriman, B. and Osher, S., "A Non-Oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (The Ghost Fluid Method)", J. Comput. Phys. 152, 457-492 (1999).

[9] Foster, N. and Metaxas, D., "Controlling Fluid Animation", Computer Graphics International 97, 178-188 (1997).

[10] Foster, N. and Metaxas, D., "Modeling the Motion of a Hot Turbulent Gas", ACM SIGGRAPH 97, 181-188 (1997).

[11] Foster, N. and Metaxas, D., "Realistic Animation of Liquids", Graphical Models and Image Processing 58, 471-483 (1996).

[12] Fournier, A. and Reeves, W.T., "A Simple Model of Ocean Waves", ACM SIGGRAPH 86, 75-84 (1986).

[13] Gates, W.F., "Interactive Flow Field Modeling for the Design and Control of Fluid Motion in Computer Animation", UBC CS Master's Thesis, 1994.

[14] Golub, G.H. and Van Loan, C.F., "Matrix Computations", The John Hopkins University Press, 1996.

[15] Harlow, F.H. and Welch, J.E., "Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with a Free Surface", The Physics of Fluids 8, 2182-2189 (1965).

[16] Kang, M., Fedkiw, R. and Liu, X.-D., "A Boundary Condition Capturing Method For Multiphase Incompressible Flow", J. Sci. Comput. 15, 323-360 (2000).

[17] Kass, M. and Miller, G., "Rapid, Stable Fluid Dynamics for Computer Graphics", ACM SIGGRAPH 90, 49-57 (1990).

[18] Lorenson, W.E. and Cline, H.E., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Computer Graphics 21, 163-169 (1987).

[19] Miller, G. and Pearce, A., "Globular Dynamics: A Connected Particle System for Animating Viscous Fluids", Computers and Graphics 13, 305-309 (1989).

[20] O'Brien, J. and Hodgins, J., "Dynamic Simulation of Splashing Fluids", Computer Animation 95, 198-205 (1995).

[21] Osher, S. and Sethian, J.A., "Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations", J. Comput. Phys. 79, 12-49 (1988).

[22] Peachy, D., "Modeling Waves and Surf", ACM SIGGRAPH 86, 65-74 (1986).

[23] Schachter, B., "Long Crested Wave Models", Computer Graphics and Image Processing 12, 187-201 (1980).

[24] Sethian, J.A. "Level Set Methods and Fast Marching Methods", Cambridge University Press, Cambridge 1999.

[25] Stam, J., "Stable Fluids", ACM SIGGRAPH 99, 121-128 (1999).

[26] Staniforth, A. and Cote, J., "Semi-Lagrangian Integration Schemes for Atmospheric Models – A Review", Monthly Weather Review 119, 2206-2223 (1991).

[27] Terzopoulos, D., Platt, J. and Fleischer, K., "Heating and Melting Deformable Models (From Goop to Glop)", Graphics Interface 89, 219-226 (1995).

[28] Yngve, G., O'Brien, J. and Hodgins, J., "Animating Explosions", ACM SIGGRAPH 00, 29-36 (2000).

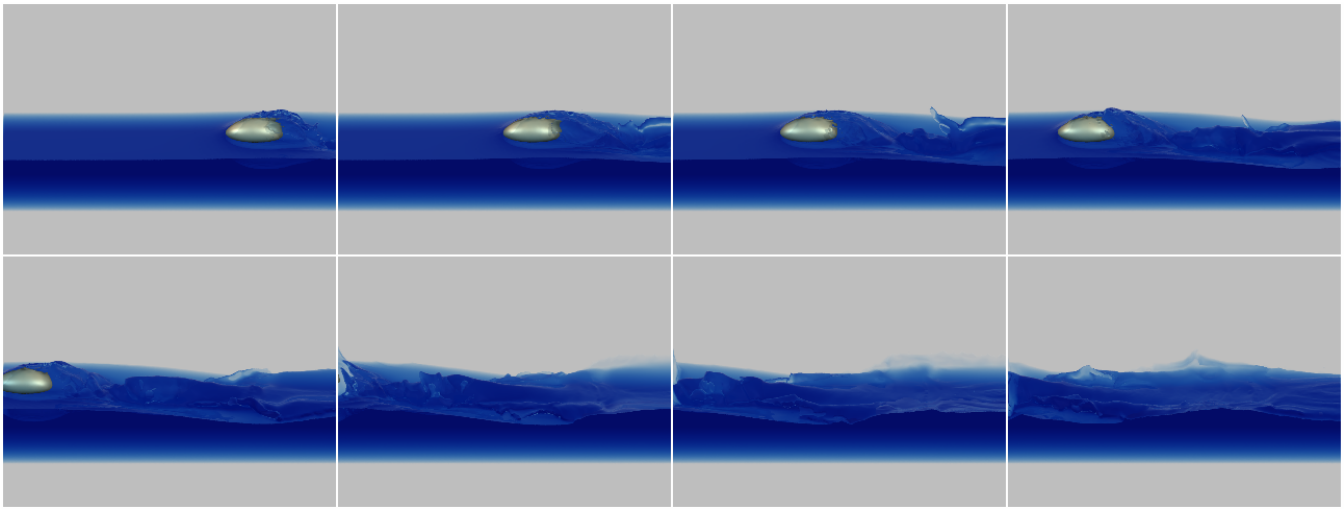


Figure 5: *An ellipsoid slips along through shallow water. The combination of particle and level set tracking allows water to flow over the object without any visual loss of volume. The environment for this example was 250x75x90 cells. It took approximately seven minutes to calculate the liquid motion (including surface evolution) per frame.*

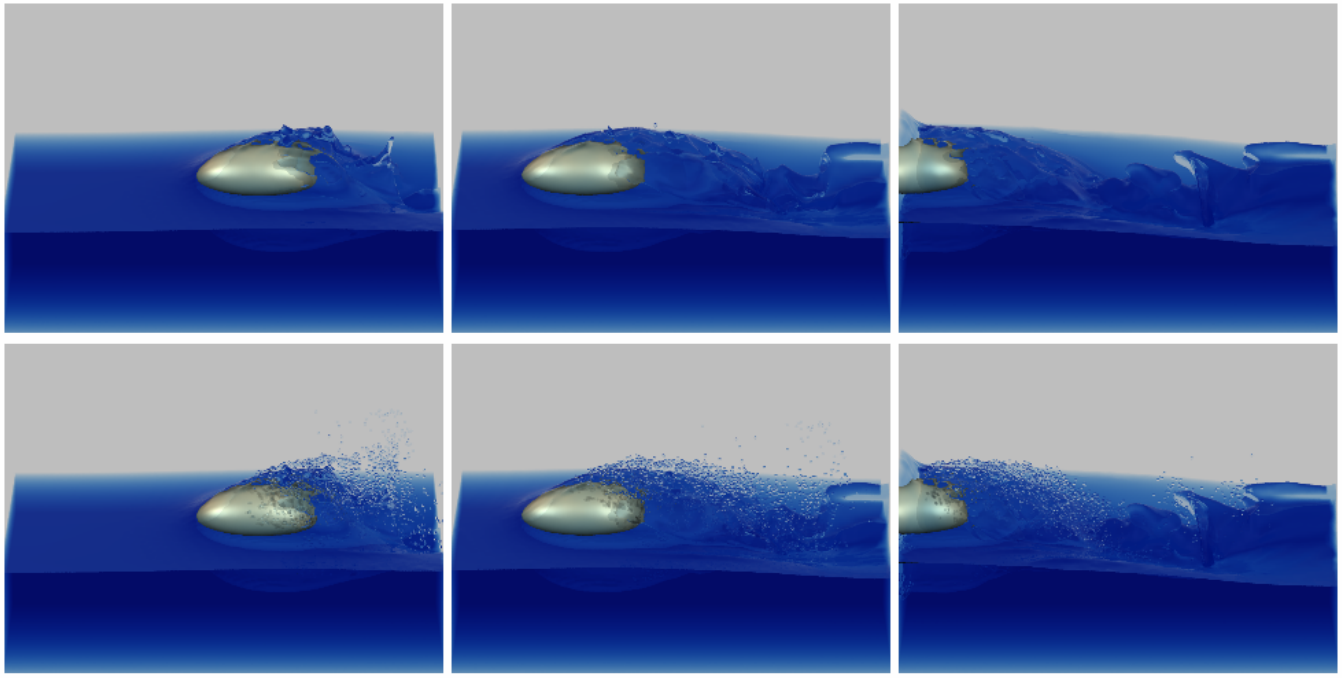


Figure 6: *A close up of the ellipsoid from figure 5 showing the implicit surface derived from combining the particle basis functions and level set (top), and with the addition of the freely splashing particles raytraced as small spheres (bottom). The environment for this example was 150x75x90 cells. Calculation times were approximately four minutes per frame.*

Figure 7: *A fully articulated animated character interacts with viscous mud. The environment surrounding the character is 150x200x150 cells. That resolution is sufficient to accurately model the character filling his mouth with mud. A 3D control curve is used to eject (spit) the mouthful of mud later in the sequence. This example runs at three minutes per frame.*

Animation and Rendering of Complex Water Surfaces

Douglas Enright
Stanford University
Industrial Light & Magic
enright@stanford.edu

Stephen Marschner
Stanford University
srm@graphics.stanford.edu

Ronald Fedkiw
Stanford University
Industrial Light & Magic
fedkiw@cs.stanford.edu

Abstract

We present a new method for the animation and rendering of *photorealistic* water effects. Our method is designed to produce visually plausible three dimensional effects, for example the pouring of water into a glass (see figure 1) and the breaking of an ocean wave, in a manner which can used in a computer animation environment. In order to better obtain photorealism in the behavior of the simulated water surface, we introduce a new "thickened" front tracking technique to accurately represent the water surface and a new velocity extrapolation method to move the surface in a smooth, water-like manner. The velocity extrapolation method allows us to provide a degree of control to the surface motion, e.g. to generate a wind-blown look or to force the water to settle quickly. To ensure that the photorealism of the simulation carries over to the final images, we have integrated our method with an advanced physically based rendering system.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing;

Keywords: computational fluid dynamics, implicit surfaces, natural phenomena, physically based animation, rendering, volume rendering

1 Introduction

Water surrounds us in our everyday lives. Given the ubiquity of water and our constant interaction with it, the animation and rendering of water poses one of the greatest challenges in computer graphics. The difficulty of this challenge was underscored recently through the use of water effects in a variety of motion pictures including the recent feature film "Shrek" where water, mud, beer and milk effects were seen. In response to a question concerning what was the single hardest shot in "Shrek", DreamWorks SKG principal and producer of "Shrek", Jeffrey Katzenberg, stated, *It's the pouring of milk into a glass.* [Hiltzik and Pham 2001].

The above quote illustrates the need for *photorealistic* simulation and rendering of water (and other liquids such as milk), especially in the case of complex, three dimensional behavior as seen when water is poured into a glass as in figure 1. A key to achieving this goal is the visually accurate treatment of the surface separating the water from the air. The behavior of this surface provides the visual impression that water is being seen. If the numerical simulation method used to model this surface is not robust enough to capture the essence of water, then the effect is ruined for the viewer. A variety of new techniques for treatment of the surface are proposed in this paper in order to provide visually pleasing motion and photorealistic results.

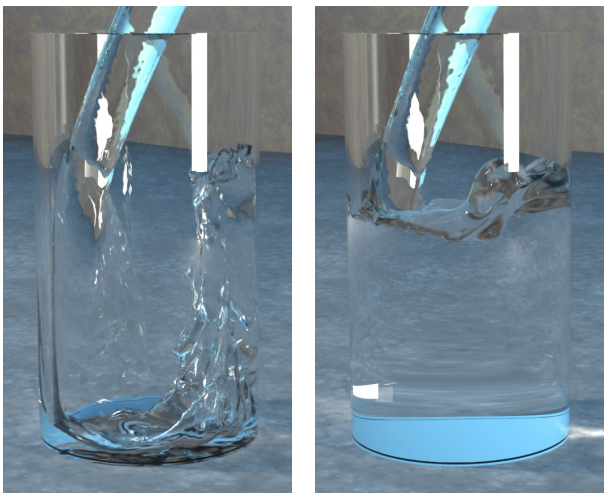


Figure 1: **Water being poured into a glass (55x120x55 grid cells).**

We propose a new "thickened" front tracking approach to modeling the surface, called the "particle level set method". It is a hybrid surface tracking method that uses massless marker particles combined with a dynamic implicit surface. This method was entirely inspired by the hybrid liquid volume model proposed in [Foster and Fedkiw 2001], but exhibits significantly more realistic surface behavior. This effect is achieved by focusing on modeling the surface as opposed to the liquid volume as was done by [Foster and Fedkiw 2001]. This shift in philosophy away from volume modeling and towards surface modeling, is the key idea behind our new techniques, resulting in better photorealistic behavior of the water surface.

We propose a new treatment of the velocity at the surface in order to obtain more visually realistic water surface behavior. The motion of both the massless marker particles and the implicit function representing the surface is dependent upon the velocities contained on the underlying computational mesh. By extrapolating velocities across the water surface and into the region occupied by the air, we obtain more accurate and better visual results. In the limit as the computational grid is refined, the resulting surface condition is identical to the traditional approach of making the velocity divergence free, but it gives more visually appealing and physically plausible results on coarse grids. Furthermore, this velocity extrapolation procedure alows us to add a degree of control to the behavior of the water surface. We can add dampening and/or churning effects forcing it to quiet down or splash up faster than would be

allowed by a straightforward physical simulation.

Our new advances can be easily incorporated into a pre-existing Navier-Stokes solver for water. In fact, we solve the Navier-Stokes equations for liquid water along the lines of [Foster and Fedkiw 2001], in particular using a semi-Lagrangian "stable fluid" approach introduced to the community by Stam [Stam 1999]. Our approach preserves as much of the realistic behavior of the water as possible, while at the same time providing a degree of control necessary for its use in a computer animation environment.

Photorealistic rendering is necessary in order to complete the computational illusion of real water. In some ways water is an easy material to render, because unlike many common materials its optical properties are well understood and are easy to describe. In all but the largest-scale scenes, surface tension prevents water surfaces from exhibiting the microscopic features that make reflection from many other materials so complicated. However, water invariably creates situations in which objects are illuminated through complex refracting surfaces, which means that the light transport problem that is so easy to state is difficult to solve. Most widely used rendering algorithms disregard this sort of illumination or handle it using simple approximations, but because water and its illumination effects are so familiar these approaches fail to achieve realism. There are several rendering algorithms that can properly account for all transport paths, including the illumination through the water surface; some examples are path tracing [Kajiya 1986], bidirectional path tracing [Heckbert 1990; Lafortune and Willems 1993; Veach and Guibas 1994], Metropolis light transport [Veach and Guibas 1997], and photon mapping [Jensen 1995]. In our renderings of clear water for this paper we have chosen photon mapping because it is simple and it makes it easy to avoid the distracting noise that often arises in pure path sampling algorithms from illumination through refracting surfaces.

2 Previous Work

Early (and continuing) work by the graphics community into the modeling of water phenomenon focused on reduced model representations of the water surface, ranging from Fourier synthesis methods [Masten et al. 1987] to parametric representations of the water surface [Schachter 1980; Fournier and Reeves 1986; Peachey 1986; Ts'o and Barsky 1987]. The last three references are notable in the way they attempt to model realistic wave behavior including the change in wave behavior as a function of the depth of the water. Fairly realistic two dimensional wave scenery can be developed using these methods including the *illusion* of breaking waves, but ultimately they are all constrained by the sinusoidal modeling assumption present in each of them. They are unable to easily deal with complex three dimensional behaviors such as flow around objects and dynamically changing boundaries. A summary of the above methods and their application to modeling and rendering of ocean wave environments can be found in [Tessendorf 2001].

In order to obtain water models which could potentially be used in a dynamic animation environment, researchers turned towards using two dimensional approximations to the full 3D Navier-Stokes equations. [Kass and Miller 1990] use a linearized form of the 2D shallow water equations to obtain a height field representation of the water surface. A pressure defined height field formulation was used by [Chen and Lobo 1994] in fluid simulations with moving obstacles. [O'Brien and Hodgins 1995] used a height model combined with a particle system in order to simulate splashing liquids. The use of a height field gives a three dimensional look to a two dimensional flow calculation, but it constrains the surface to be a function, i.e. the surface passes the vertical line test where for each (x,y) position there is at most one z value. The surface of a crashing wave or of water being poured into a glass does not satisfy the vertical line test. Use of particle systems permits the surface to become multivalued. A viscous spring particle representation of a liquid has been proposed by [Miller and Pearce 1989]. An alternative molecular dynamics approach to the simulation of particles in the liquid phase has been developed by [Terzopoulos et al. 1989]. Particle methods, while quite versatile, can pose difficulties when trying to reconstruct a smooth water surface from the locations of the particles alone.

The simulation of complex water effects using the full 3D Navier-Stokes equations has been based upon the large amount of research done by the computational fluid dynamics community over the past 50 years. [Foster and Metaxas 1996] utilized the work of [Harlow and Welch 1965] in developing a 3D Navier-Stokes methodology for the realistic animation of liquids. Further CFD enhancements to the traditional marker and cell method of Harlow and Welch which allow one to place particles only near the surface can be found in [Chen et al. 1997]. A semi-Lagrangian "stable fluids" treatment of the convection portion of the Navier-Stokes equations was introduced to the computer graphics community by [Stam 1999] in order to allow the use of significantly larger time steps without hindering stability. [Foster and Fedkiw 2001] made significant contributions to the simulation and control of three dimensional fluid simulations through the introduction of a hybrid liquid volume model combining implicit surfaces and massless marker particles; the formulation of plausible boundary conditions for moving objects in a liquid; the use of an efficient iterative method to solve for the pressure; and a time step subcycling scheme for the particle and implicit surface evolution equations in order to reduce the amount of visual error inherent to the large semi-Lagrangian "stable fluid" time step used for time evolving the fluid velocity and the pressure. The combination of all of the above advances in 3D fluid simulation technology along with ever increasing computational resources has set the stage for the inclusion of fully 3D fluid animation tools in a production environment.

Most work on simulating water at small scales has not specifically addressed rendering and has not used methods that correctly account for all significant light transport paths. Research on the rendering of illumination through water [Watt 1990; Nishita and Nakamae 1994] has used methods based on processing each polygon of a mesh that represents a fairly smooth water surface, so these methods cannot be used for the very complex implicit surfaces that result from our simulations. For the case of 2D wave fields in the open ocean, approaches motivated by physical correctness have produced excellent results [Premoze and Ashikhmin 2000; Tessendorf 2001].

3 Simulation Method

3.1 Motivation

[Foster and Fedkiw 2001] chose to define the liquid volume being simulated as one side of an isocontour of an implicit function, ϕ. The surface of the water was defined by the $\phi = 0$ isocontour with $\phi \leq 0$ representing the water and $\phi > 0$ representing the air. By using an implicit function representation of the liquid volume, they obtained a smooth, temporally coherent liquid surface. They rejected the use of particles alone to represent the liquid surface because it is difficult to a calculate a visually desirable smooth liquid surface from the discrete particles alone. The implicit surface was dynamically evolved in space and time according to the underlying liquid velocity $\vec{u}$. As shown in [Osher and Sethian 1988], the appropriate equation to do this is

$$\phi_t + \vec{u} \cdot \nabla \phi = 0 \tag{1}$$

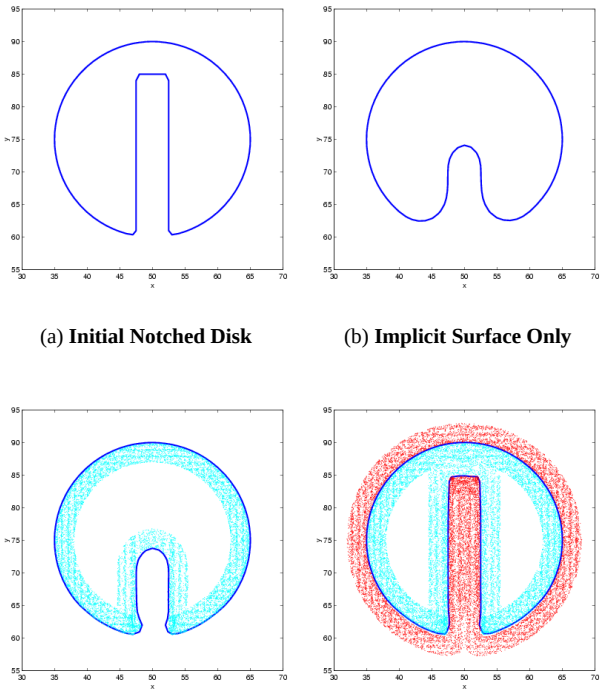
where ϕ_t is the partial derivative of ϕ with respect to time and $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$ is the gradient operator.

An implicit function only approach to modeling the surface will not yield realistic surface behavior due to an excessive amount of volume loss on coarse grids. A seminal advance of [Foster and Fedkiw 2001] in creating realistic liquids for computer animation is the hybridization of the visually pleasing smooth implicit function modeling of the liquid volume with particles that can maintain the liquid volume on coarse grids. The inclusion of particles provides a way for capturing the liveliness of a real liquid with spray and splashing effects. Curvature was used as an indicator for allowing particles to influence the implicit surface representation of the water. This is a natural choice since small droplets of water have very high curvature and dynamic implict surfaces have difficulty resolving features with sharp corners.

Figure 2(a) shows a notched disk that we rotate for one rigid body rotation about the point (50,50). The inside of the disk can be thought of as a volume of liquid. Figure 2(b) shows the result of using an implicit surface only approach (after one rotation) where both the higher and lower corners of the disk are erroneously shaved off causing both loss of visual features and an artificially viscous look for the liquid. This numerical result was obtained using a highly accurate fifth order WENO discretization of equation 1 (see e.g. [Jiang and Peng 2000; Osher and Fedkiw 2002]). For the sake of comparison, we note that [Sethian 1999] only proposes second order accurate methods for discretizing this equation. Figure 2(c) shows the result obtained with our implementation of the method from [Foster and Fedkiw 2001]. The particles inside the disk do not allow the implicit surface to cross over them and help to preserve the two corners near the bottom. However, there is little they can do to stop the implicit surface from drifting away from them near the top corners. This represents loss of air or bubbles as the method erroneously gains liquid volume. This is not desirable since many complex water motions such as wave phenomenon are due in part to differing heights of water columns adjacent to each other. Loss of air in a water column reduces the pressure forces from neighboring columns destroying many of the dynamic splashing effects as well as the overall visually stimulating liveliness of the liquid.

While the hybrid liquid volume model of Foster and Fedkiw attempts to maintain the volume of the liquid accurately, it fails to model the air or more generally the opposite side of the liquid surface. We shift the focus away from maintaining a liquid volume towards maintaining the liquid surface itself. An immediate advantage of this approach is that it leads to symmetry in particle placement. We place particles on both sides of the surface and use them to maintain an accurate representation of the surface itself regardless of what may be on one side or the other. The particles are not meant to track volume, they are intended to correct errors in the surface representation by the implicit function. In [Enright et al. 2002] we showed that this surface only approach leads to the most accurate 3D results for surface tracking ever achieved (in both CFD and CG). This was done for analytical "test" velocity fields. Figure 2(d) shows that this new method correctly computes the rigid body rotation for the notched disk preserving both the water and the air volumes so that more realistic water motion can be obtained.

In this current paper, we couple this new method to real velocity fields and fluid dynamics calculations for the first time. Representative results of this new method can be seen in figure 3. A ball is thrown into a tank of water with the same tank geometry, grid spacing and ball speed as seen in figure 4 (courtesy of [Foster and Fedkiw 2001]). The resulting splash after the ball impacts the surface of the water is dramatically different between the two figures. Our new method produces the well formed, thin sheet one would visually expect. Note that the distorted look of the ball in our figure is due to the correct calculation of the refraction of light when it passes through the surface of the water. To give an indication of the additional computational cost incurred using our new method,



(a) **Initial Notched Disk** (b) **Implicit Surface Only**

(c) **Particles Inside Only** (d) **Our New Method**

Figure 2: **Rigid Body Rotation Test**

figure 3 took about 11 minutes per frame, figure 4 took about 7 minutes per frame and a level set only solution takes about 3 minutes per frame.

3.2 Particle Level Set Method

3.2.1 Initialization of Particles

Two sets of particles are randomly placed in a "thickened" surface region (we use three grid cells on each side of the surface) with *positive* particles in the $\phi > 0$ region and *negative* particles in the $\phi \leq 0$ region. There is no need to place particles far away from the surface since the sign of the level set function readily identifies these regions gaining large computational savings. The number of particles placed in each cell is an adjustable parameter that can be used to control the amount of resolution, e.g. we use 64 partlces per cell for most of our examples. Each particle possesses a radius, r_p, which is constrained to take a minimum and maximum value based upon the size of the underlying computational cells used in the simulation. A minimum radius of $.1\min(\Delta x, \Delta y, \Delta z)$ and maximum radius of $.5\min(\Delta x, \Delta y, \Delta z)$ appear to work well. The radius of a particle changes dynamically throughout the simulation, since a particle's location relative to the surface changes. The radius is set according to:

$$r_p = \begin{cases} r_{max} & \text{if } s_p\phi(\vec{x}_p) > r_{max} \\ s_p\phi(\vec{x}_p) & \text{if } r_{min} \leq s_p\phi(\vec{x}_p) \leq r_{max} \\ r_{min} & \text{if } s_p\phi(\vec{x}_p) < r_{min} \end{cases}, \qquad (2)$$

where s_p is the sign of the particle (+1 for positive particles and -1 for negative particles). This radius adjustment keeps the boundary of the spherical particle tangent to the surface whenever possible. This fact combined with the overlapping nature of the particle spheres allows for an enhanced reconstruction capability of the liquid surface.

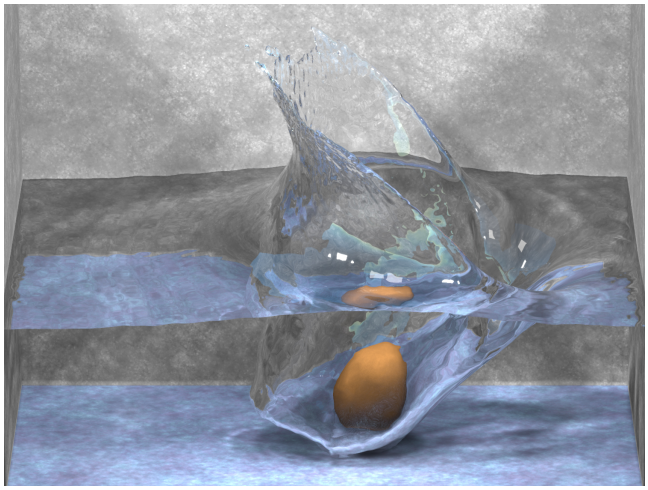


Figure 3: **Our New Method (140x110x90 grid cells).**

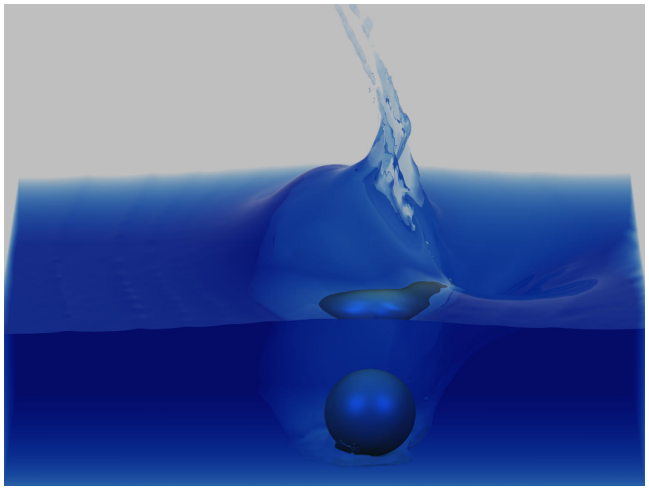


Figure 4: **Foster and Fedkiw 2001 (140x110x90 grid cells).**

3.2.2 Time Integration

The marker particles and the implicit function are separately integrated forward in time using a forward Euler time integration scheme. The implicit function is integrated forward using equation 1, while the particles are passively advected with the flow using $d\vec{x}_p/dt = \vec{u}_p$, where $\vec{u}_p$ is the fluid velocity interpolated to the particle position $\vec{x}_p$.

3.2.3 Error Correction of the Implicit Surface

Identification of Error: The main role of the particles is to detect when the implicit surface has suffered inaccuracies due to the coarseness of the computational grid in regions with sharp features. Particles that are on the wrong side of the interface by more than their radius, as determined by a locally interpolated value of ϕ at the particle position $\vec{x}_p$, are considered to have *escaped* their side of the interface. This indicates errors in the implicit surface representation. In smooth, well resolved regions of the interface, our dynamic implicit surface is highly accurate and particles do not drift a non-trivial distance across the interface.

Quantification of Error: We associate a spherical implicit function , designated ϕ_p, with each particle p whose size is determined by the particle radius, i.e.

$$\phi_p(\vec{x}) = s_p(r_p - |\vec{x} - \vec{x}_p|). \tag{3}$$

Any difference in ϕ from ϕ_p indicates errors in the implicit function representation of the surface. That is, the implicit version of the surface and the particle version of the surface disagree.

Error Correction: We use escaped positive particles to rebuild the $\phi > 0$ region and escaped negative particles to rebuild the $\phi \leq 0$ region as defined by the implicit function. The reconstruction of the implicit surface occurs locally within the cell that each escaped particle currently occupies. Using equation 3, the ϕ_p values of escaped particles are calculated for the eight grid points on the boundary of the cell containing the particle. This value is compared to the current value of ϕ for each grid point and we take the smaller value (in magnitude) which is the value closest to the $\phi = 0$ isocontour defining the surface. We do this for all escaped positive and escaped negative particles. The result is an improved representation of the surface of the liquid.

3.2.4 When To Apply Error Correction

We apply the error correction method discussed above after any computational step in which ϕ has been modified in some way. This occurs when ϕ is integrated forward in time and when the implicit function is smoothed to obtain a visually pleasing surface. We smooth the implicit surface with an equation of the form

$$\phi_\tau = -S(\phi_{\tau=0})(|\nabla\phi| - 1), \tag{4}$$

where τ is a fictitious time and $S(\phi)$ is a smoothed signed distance function given by

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + (\Delta x)^2}}. \tag{5}$$

More details on this are given in [Foster and Fedkiw 2001].

3.2.5 Particle Reseeding

In complex flows, a liquid interface can be stretched and torn in a dynamic fashion. The use of only an initial seeding of particles will not capture these effects well, as regions will form that lack a sufficient number of particles to adequately perform the error correction step. Periodically, e.g. every 20 frames, we randomly reseed particles about the "thickened" interface to avoid this dilema. This is done by randomly placing particles near the interface, and then using geometric information contained within the implicit function (e.g. the direction of the shortest possible path to the surface is given by $\vec{N} = \nabla\phi/|\nabla\phi|$) to move the particles to their respective domains, $\phi > 0$ or $\phi \leq 0$. The goal of this reseeding step is to preserve the initial particle resolution of the interface, e.g. 64 particles per cell. Thus, if a given cell has too few or too many particles, some can be added or deleted respectively.

3.2.6 A Note on Alternative Methods

If we felt that preserving the volume of the fluid was absolutely necessary in order to obtain visually pleasing fluid behavior, we would have chosen to use a volume of fluid (VOF) [Hirt and Nichols 1981] representation of the fluid. Although VOF methods explicitly conserve volume, they produce visually disturbing artifacts allowing thin liquid sheets to artificially break up and form "blobbies" and "flotsam" of liquid. Also, a visually desirable smooth fluid interface is difficult to construct when using these methods.

Another alternative is to explicitly discretize the free surface with particles and maintain a connectivity list between particles, see e.g. [Unverdi and Tryggvason 1992]. This connectivity list is difficult to maintain when parts of the free surface break apart or merge together as is often seen in complex flows of water and other liquids. Our approach avoids the especially difficult issues associated with maintaining particle connectivity information.

3.3 Velocities at the Surface

Although the Navier-Stokes equations can be used to find the velocity within the liquid volume, boundary conditions are needed for the velocity on the air side near the free surface. These boundary condition velocities are used in updating the Navier-Stokes equations, moving the surface, and moving the particles placed near the surface. The velocity at the free surface of the water can be determined through the usual enforcement of the conservation of mass (volume) constraint, $\nabla \cdot \vec{u} = 0$, where $\vec{u} = (u, v, w)$ is the velocity of the liquid. This equation allows us to determine the velocities on all the faces of computational cells that contain the $\phi = 0$ isocontour. Unfortunately, the procedure for doing this is not unique when more than one face of a cell needs a boundary condition velocity. A variety of methods have been proposed, e.g. see [Chen et al. 1995] and [Foster and Metaxas 1996].

We propose a different approach altogether, the extrapolation of the velocity across the surface into the surrounding air. As the computational grid is refined, this method is equivalent to the usual method, but it gives a smoother and more visually pleasing motion of the surface on coarser (practical sized) grids. We extrapolate the velocity out a few grid cells into the air, obtaining boundary condition velocities in a band of cells on the air side of the surface. This allows us to use higher order accurate methods and obtain better results when moving the implicit surface using equation 1 and also provides velocities for updating the position of particles on the air side of the surface. Velocity extrapolation also assists in the implementation of the semi-Lagrangian "stable fluid" method, since there are times when characteristics generated by this approach look back across the interface (a number of cells) into the air region for valid velocities.

3.3.1 Extrapolation Method

The equation modeling this extrapolation for the x component of the velocity, u, is given by

$$\frac{\partial u}{\partial \tau} = -\vec{N} \cdot \nabla u, \tag{6}$$

where $\vec{N} = (n_x, n_y, n_z)$ is a unit vector perpendicular to the implicit surface and τ is fictitious time. A similar equation holds for the v and w components of velocity field. Since we use an implicit surface to describe the fluid, $\vec{N} = \nabla\phi / |\nabla\phi|$. Fast methods exist for solving this equation in $O(n \log n)$ time, where n is the number of grid points that one needs to extrapolate over, in our case a five grid cell thick band on the air side of the interface. The fast method is based upon the observation that information in equation 6 propagates in only one direction away from the surface. This implies that we do not have to revisit previously computed values of $\vec{u}_{ext}$ (the extrapolated velocity) if we perform the calculation in the correct order. The order is determined by the value of ϕ allowing us to do an $O(n \log n)$ sorting of the points before beginning the calculation. The value of u itself is determined by enforcing the condition at steady state, namely $\nabla\phi \cdot \nabla u = 0$ where the derivatives are determined using previously calculated values of ϕ and u. From this scalar equation, a new value of u can be determined, and then we proceed to the next point corresponding to the next smallest value of ϕ, etc. Further details of this method are discussed in [Adalsteinsson and Sethian 1999].

3.3.2 Velocity Advection

The momentum portion of the Navier-Stokes equations is:

$$\vec{u}_t = -\vec{u} \cdot \nabla\vec{u} + \nu\nabla \cdot (\nabla\vec{u}) - \frac{1}{\rho}\nabla p + \vec{g}, \tag{7}$$

where ν is the kinematic viscosity of the fluid, ρ is the density of the fluid, p is the pressure, and $\vec{g}$ is an externally applied gravity field. We use the semi-Lagrangian "stable fluids" method [Stam 1999] to update the convective portion of this equation, i.e. the $\vec{u} \cdot \nabla\vec{u}$ term. This method calculates the first term on the left hand side of equation 7 by following the fluid characteristics backwards in time to determine from which computational cell the volume of fluid came, and then taking an average of the appropriate velocities there. This allows one to stably take much larger time steps than would be allowed using other time advancement schemes for velocity advection. A consequence of the now allowed large semi-Lagrangian time step is that near the surface, we might look across the interface as many as a few grid cells into the air region to find velocities. In a standard approach, valid velocities are not defined in this region. However, our velocity extrapolation technique easily handles this case ensuring that physically plausible velocities exist a few grid cells into the air region. In fact, we extrapolate the velocity the maximum distance from the surface that would be allowed during a semi-Lagrangian "stable fluids" time step guaranteeing that a smooth, physically plausible and visually appealing velocity can be found there.

3.3.3 Control

Our velocity extrapolation method enables us to apply a newly devised method for controlling the nature of the surface motion. This is done simply by modifying the extrapolated velocities on the air side of the surface. For example, to model wind-blown water as a result of air drag, we take a convex combination of the extrapolated velocities with a pre-determined wind velocity field

$$\vec{u} = (1 - \alpha)\vec{u}_{ext} + \alpha\vec{u}_{wind}, \tag{8}$$

where $\vec{u}_{ext}$ is the extrapolated velocity, $\vec{u}_{wind}$ a desired air-like velocity, and $0 \leq \alpha \leq 1$ the mixing constant. This can be applied throughtout the surface or only locally in select portions of the computational domain as desired. Note that setting $\vec{u}_{wind} = 0$ forces churning water to settle down faster with the fastest settling resulting from $\alpha = 1$. All of the figures shown in the paper used $\alpha = 0$, but we demonstrate how α can be used to force a poured glass of water to settle more quickly in the accompanying video.

3.4 Summary

We divide up the computational domain into voxels with the components of $\vec{u}$ stored on the appropriate faces and p, ρ and ϕ stored at the center of each cell. This arrangement of computational variables is the classic staggered MAC-style arrangement [Harlow and Welch 1965]. The density of a given cell is given by the value of ϕ at the center of the cell. The evolution of $\vec{u}$, p, ρ and ϕ in a given time step is performed in a series of three steps as outlined below:

1. The current surface velocity is smoothly extrapolated across the surface into the air region as discussed in section 3.3.1. Appropriate control behavior modifications to the velocity field are made.

2. The water surface and marker particles are integrated forward in time via an explicit time step subcycling method with the appropriate corrections to ϕ as described in section 3.2.3.

3. The velocity field is updated with equation 7 using the updated values for ρ. This is done by first using the semi-Lagrangian "stable fluid" method to find an estimate for the velocity. This estimate is further augmented by the viscous and forcing terms. The spatial derivatives used in calculating these terms are calculated using a standard centered second order accurate finite difference scheme. Then a system of linear equations is assembled and solved for the pressure in order to make this intermediate velocity field divergence free. The newly calculated pressure is applied as a correction to the intermediate velocity in order to fully update the water's velocity field. Interaction of the liquid with objects, walls, etc. is treated here as well. For this step, we follow exactly the method of [Foster and Fedkiw 2001] and refer the reader there for more details.

This sequence of steps is repeated until a user defined stopping point is reached. The time step for each iteration of the above steps is determined using the water's velocity to calculate an appropriate CFL condition which is approximately five times larger than the traditional CFL condition used in fluid simulations (the semi-Lagrangian "stable fluids" method allows this). Also, any viscosity related CFL restrictions are locally dealt with by reducing the viscosity in the offending cells in order to allow for our larger time step.

3.5 Rendering

Our results are rendered using a physically based Monte Carlo ray tracer capable of handling all types of illumination using photon maps and irradiance caching [Jensen 2001]. To integrate our simulation system with the renderer we implemented a geometry primitive that intersects rays with the implicit surface directly by solving for the root of the signed distance along the ray. Depending on the accuracy required by a particular scene we use either a trilinear or a tricubic filter [Marschner and Lobb 1994] to reconstruct ϕ. The normal is computed using trilinearly interpolated central differences for the trilinear surface, or simply using the derivative of the reconstructed tricubic surface.

The properties of ϕ have two advantages in the rendering context. First, intersecting a ray with the surface is much more efficient than it would be for an isosurface of a general function. The signed distance function provides a lower bound on the distance to the intersection along the ray, allowing us to take large steps when the current point is far from the surface. Second, it is easy to provide for motion blur in the standard distribution ray tracing framework. To compute the surface at an intermediate time between two frames we simply interpolate between the two signed distance volumes and use the same intersection algorithm unchanged. For the small motions that occur between frames the special properties of ϕ are not significantly compromised.

4 Results

4.1 Pouring Water Into A Glass

Figures 1 and 7 show the high degree of complexity in the water surface. Note the liveliness of the surface of the water when the water is initially being poured. The ability to maintain the visually pleasing thin sheets of water during the turbulent mixing phase is a consequence of our new method. We do not loose any of the fine detail with regards to the air pockets formed, since we model both sides of the water surface. Even though the calculation was performed on a Cartesian computational grid, the glass was shaped as a cylinder on the grid, with the grid points outside the cylinder treated as an object which does not permit the fluid to interpenetrate it. The glass was modeled as smooth and clear in order to highlight the action of the water being poured into the glass. The computational cost was approximately 8 minutes per frame.

To our knowledge, the only other complex, three dimensional simulation of a liquid being poured into a glass for computer animation is from the Gingerbread Man torture scene in the feature film "Shrek". Milk lacks the transparency of water making it difficult to clearly view the dynamic behavior of the milk surface. Also, the modeling of a thick polygonal glass with a rough surface does not provide a clear view of the milk making a direct comparison difficult.

The scene used to render our result contains just a simple cylindrical glass, the simulated water surface, and a few texture mapped polygons for the background. Illumination comes from a physically based sky model and two area sources. Because water only reflects and refracts other objects, the scene including the sky is very important to the appearance. The glass and the water together create three dielectric interfaces: glass-air, water-air, and air-water, so the inside surface of the glass has two sets of material properties depending on whether it is inside or outside the implicitly defined surface (which is easy to determine by checking the sign of ϕ). The illumination in the shadow of the glass is provided by a photon map, and illumination from the sky on diffuse surfaces in the scene is computed using Monte Carlo integration. We also note that milk would not present as difficult a global illumination problem due to its lack of transparency. Motion blur was included for these renderings.

4.2 Breaking Wave

As a second example, we have performed a breaking wave calculation in a numerical wave tank as shown in figure 8. To begin to model this pheonmenon, we needed to chose an initial condition for the wave. We chose to use the theoretical solution of a solitary wave of finite amplitude propagating without shape change [Radovitzky and Ortiz 1998]. The initial velocities u and v in the x and y directions respectively and surface height η are given by:

$$u = \sqrt{gd}\frac{H}{d}\operatorname{sech}^2\left[\sqrt{\frac{3H}{4d^3}}x\right] \quad (9)$$

$$v = \sqrt{3gd}\left(\frac{H}{d}\right)^{3/2}\frac{y}{d}\operatorname{sech}^2\left[\sqrt{\frac{3H}{4d^3}}x\right]\tanh\left[\sqrt{\frac{3H}{4d^3}}x\right] \quad (10)$$

$$\eta = d + H\operatorname{sech}^2\left[\sqrt{\frac{3H}{4d^3}}x\right], \quad (11)$$

where g is the gravitational constant 9.8 m/s. For our simulations, we set $H = 7$ and $d = 10$. We used the same initial conditions in three spatial dimensions, replicating the two dimensional initial conditions along the z-axis.

Next, we needed to introduce a model of a sloping underwater shelf in order cause the propagating pulse to actually pile up on itself and form a breaking wave. We performed a variety of two dimensional tests to determine the best underwater shelf geometry to generate a visually pleasing breaking wave. Figure 5 is a sequence of frames from one of the two dimensional tests we ran with the same x-y cross section as can be found in our 3D example. We found this prototyping technique to be a fast and easy way to explore possible breaking wave behaviors in a fraction of the time it would take to run a fully three dimensional test case.

After determining the best submerged shelf geometry to generate a breaking wave, we generated a slight tilt in the geometry in order

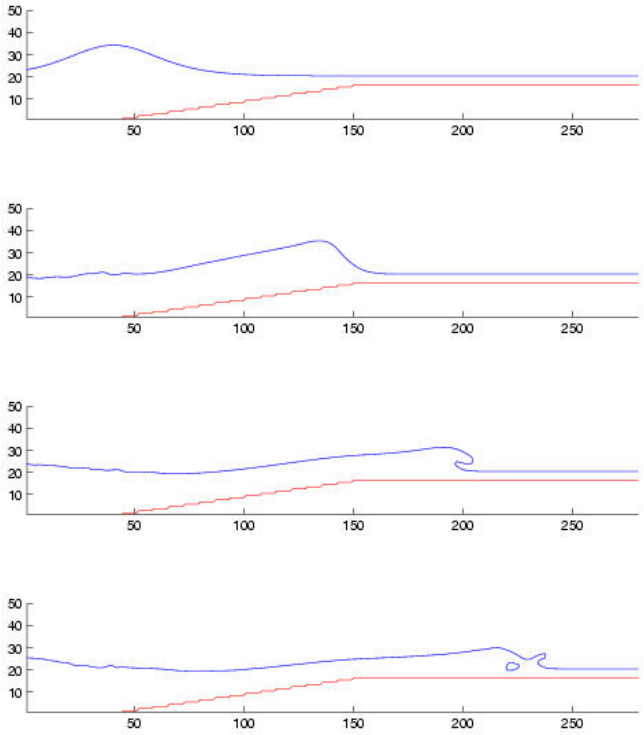


Figure 5: **Two Dimensional Breaking Wave**

to induce a curl in the break of the wave and enhance the three dimensional look. A sketch of the shelf geometry used in our three dimensional calculations is shown in figure 6. An incline of slope 1:7 rises up from the seabottom to a depth of 2 m below the surface of the water. Instead of allowing the incline to go all the way to the surface we chose to have it flatten out at this depth in order to illustrate the splash up effects after the initial breaking of the wave.

Next we ran a fully three dimensional simulation, the results of which can be seen in figures 8 and 9. The wave has the intended curling effect. The formation of a tube of air is clearly seen after the wave splashes down, with the "air" particles maintaining the tube even after the wave begins a secondary splash up. We observe some solely three dimensional effects including the fingering of the breaking wave. The computational cost was approximately 3 minutes per frame.

Because of the very different scale of this simulation as compared to the water glass, different optical effects are important. We model the water volume under the surface, which in surf is densely populated with scattering particles, as a gray diffuse reflector. The color of the water comes from a cloudy blue sky. Because the features that can be resolved are so fine compared to the grid resolution, we used cubic interpolation for this scene.

The simulation shown captures the basic phenomena of the breaking wave on a very coarse grid, but in real waves there are small-scale features that, while not important to the *behavior* of the large wave, are very important for its *appearance*. Coupling a 2D simulation for the small scale features to the wave simulation is a promising avenue for future work. Once those waves are incorporated it will become important to treat diffusion of light through the water more correctly. Also, no texture mapping was performed, e.g. use of a Philips spectrum [Tessendorf 2001], or bump mapping technique [Fournier and Reeves 1986]. Another challenge will be to augment our method to naturally handle spray and foam.

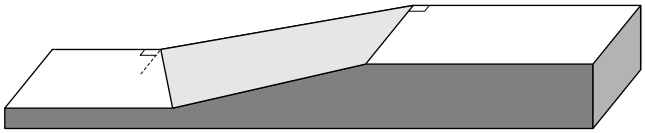


Figure 6: **Submerged Shelf**

5 Conclusion and Future Work

We have presented some novel computational methods for enhanced surface tracking and modeling of the surface motion. Combining these leads to the possibility of creating photorealistic behavior in 3D water simulations for the purpose of computer graphics animation. We discussed some advances in rendering the simulated photorealistic behavior in order to complete the illusion that real water is being seen. The computational methods presented can easily be included into an existing three dimensional fluid simulation animation tool. While we have not done any texture mapping of the water surfaces, we believe that our "thickened" front tracking approach should enable texture mapping of an implicitly defined fluid surface, and we will pursue this as future work.

6 Acknowledgment

Research supported in part by an ONR YIP and PECASE award N00014-01-1-0620, NSF DMS-0106694, NSF ACI-0121288, NSF IIS-0085864, and the DOE ASCI Academic Strategic Alliances Program (LLNL contract B341491).

The authors acknowledge Henrik Wann Jensen for the use of his *dali* rendering system and for his help with the extensions that were added to it for this paper.

The third author would like to acknowledge the fruitful collaboration with Nick Foster regarding the hybridization of particle and level set methods that was published in [Foster and Fedkiw 2001]. Without it, neither [Enright et al. 2002] nor this current work would have been possible.

References

ADALSTEINSSON, D., AND SETHIAN, J. 1999. The fast construction of extension velocities in level set methods. *J. Comp. Phys. 148*, 2–22.

CHEN, J., AND LOBO, N. 1994. Toward interactive-rate simulation of fluids with moving obstacles using the navier-stokes equations. *Computer Graphics and Image Processing 57*, 107–116.

CHEN, S., JOHNSON, D., AND RAAD, P. 1995. Velocity boundary conditions for the simulation of free surface fluid flow. *J. Comp. Phys. 116*, 262–276.

CHEN, S., JOHNSON, D., RAAD, P., AND FADDA, D. 1997. The surface marker and micro cell method. *Int. J. for Num. Meth. in Fluids 25*, 749–778.

ENRIGHT, D., FEDKIW, R., FERZIGER, J., AND MITCHELL, I. 2002. A hybrid particle level set method for improved interface capturing. *J. Comp. Phys.*.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM

SIGGRAPH, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graphical Models and Image Processing 58*, 471–483.

FOURNIER, A., AND REEVES, W. T. 1986. A simple model of ocean waves. In *Computer Graphics (Proceedings of SIGGRAPH 86), 20(4)*, ACM, 75–84.

HARLOW, F., AND WELCH, J. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids 8*, 2182–2189.

HECKBERT, P. S. 1990. Adaptive radiosity textures for bidirectional ray tracing. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, ACM, 145–154.

HILTZIK, M. A., AND PHAM, A. 2001. Synthetic actors guild. *Los Angeles Times*. May 8, 2001, natl. ed. : A1+.

HIRT, C., AND NICHOLS, B. 1981. Volume of fluid (VOF) method for the dynamics of free boundaries. *J. Comp. Phys. 39*, 201–225.

JENSEN, H. W. 1995. Importance driven path tracing using the photon map. In *Eurographics Rendering Workshop 1995*, 326–335.

JENSEN, H. W. 2001. *Realistic Image Synthesis Using Photon Maps*. A K Peters.

JIANG, G.-S., AND PENG, D. 2000. Weighted eno schemes for hamilton-jacobi equations. *SIAM J. Sci. Comput. 21*, 2126–2143.

KAJIYA, J. T. 1986. The rendering equation. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, vol. 20, 143–150.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, ACM, 49–57.

LAFORTUNE, E. P., AND WILLEMS, Y. D. 1993. Bi-directional path tracing. In *Proceedings of Compugraphics '93*, 145–153.

MARSCHNER, S., AND LOBB, R. 1994. An evaluation of reconstruction filters for volume rendering. In *Proceedings of Visualization 94*, IEEE Comput. Soc. Press, 100–107.

MASTEN, G., WATTERBERG, P., AND MAREDA, I. 1987. Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Application 7*, 16–23.

MILLER, G., AND PEARCE, A. 1989. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics 13*, 3, 305–309.

NISHITA, T., AND NAKAMAE, E. 1994. Method of displaying optical effects within water using accumulation buffer. In *Proceedings of SIGGRAPH 94*, ACM SIGGRAPH / ACM Press, Computer Graphics Proceedings, Annual Conference Series, ACM, 373–381.

O'BRIEN, J., AND HODGINS, J. 1995. Dynamic simulation of splashing fluids. In *Proceedings of Computer Animation 95*, 198–205.

OSHER, S., AND FEDKIW, R. 2002. *The Level Set Method and Dynamic Implicit Surfaces*. Springer-Verlag, New York.

OSHER, S., AND SETHIAN, J. 1988. Fronts propagating with curvature dependent speed: Algorithms based on hamiliton-jacobi formulations. *J. Comp. Phys. 79*, 12–49.

PEACHEY, D. R. 1986. Modeling waves and surf. In *Computer Graphics (Proceedings of SIGGRAPH 86), 20(4)*, ACM, 65–74.

PREMOZE, S., AND ASHIKHMIN, M. 2000. Rendering natural waters. In *The proceedings of Pacific Graphics 2000*, 23–30.

RADOVITZKY, R., AND ORTIZ, M. 1998. Lagrangian finite element analysis of newtonian fluid flows. *Int. J. Numer. Meth. Engng. 43*, 607–619.

SCHACHTER, B. 1980. Long crested wave models. *Computer Graphics and Image Processing 12*, 187–201.

SETHIAN, J. 1999. *Level Set Methods and Fast Marching Methods*. Cambridge University Press.

STAM, J. 1999. Stable fluids. In *Proceedings of SIGGRAPH 99*, ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, ACM, 121–128.

TERZOPOULOS, D., PLATT, J., AND FLEISCHER, K. 1989. Heating and melting deformable models (from goop to glop). In *Graphics Interface '89*, 219–226.

TESSENDORF, J. 2001. Simulating ocean water. In *SIGGRAPH 2001 Course Notes*.

TS'O, P. Y., AND BARSKY, B. A. 1987. Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Transactions on Graphics 6*, 3, 191–214.

UNVERDI, S.-O., AND TRYGGVASON, G. 1992. A front-tracking method for viscous, incompressible, multi-fluid flows. *J. Comp. Phys. 100*, 25–37.

VEACH, E., AND GUIBAS, L. J. 1994. Bidirectional estimators for light transport. In *Eurographics Rendering Workshop 1994 Proceedings*, 147–162.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *Proceedings of SIGGRAPH 97*, ACM SIGGRAPH / Addison Wesley, Computer Graphics Proceedings, Annual Conference Series, ACM, 65–76.

WATT, M. 1990. Light-water interaction using backward beam tracing. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, ACM, 377–385.

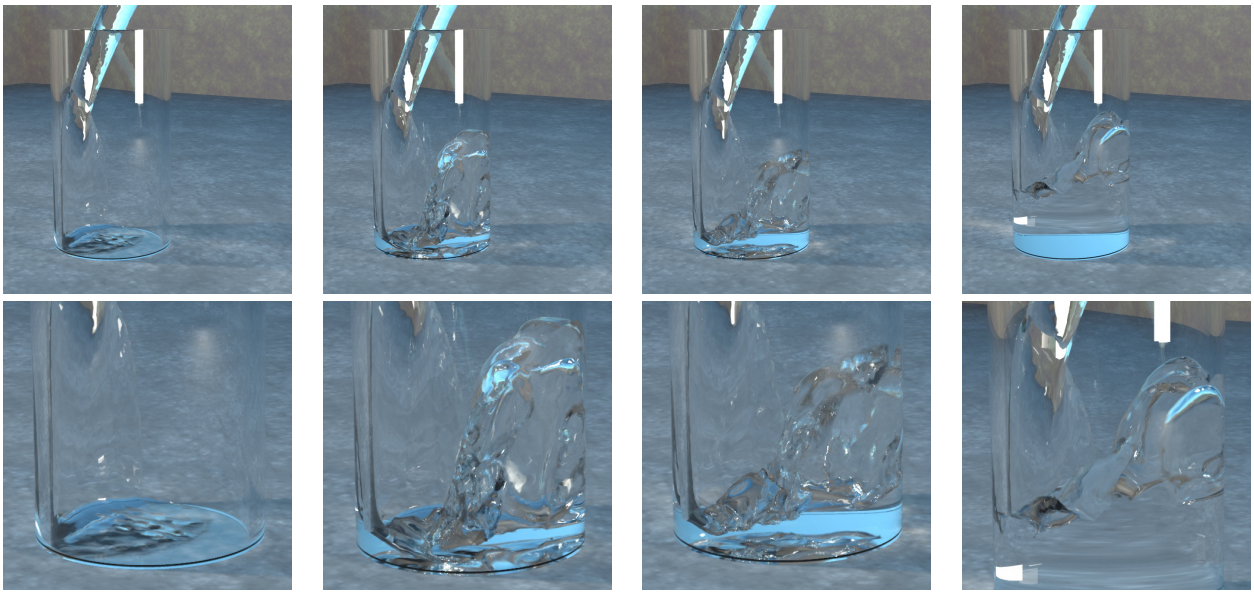


Figure 7: **Water being poured into a clear, cylindrical glass (55x55x120 grid cells). Our method makes possible the fine detail seen in the turbulent mixing of the water and air.**

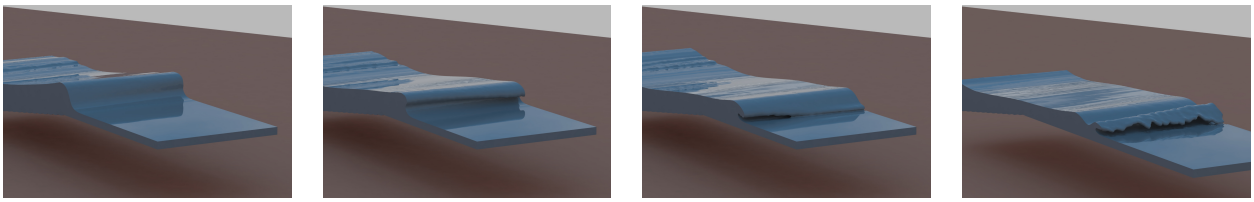


Figure 8: **Abstract view of wave breaking on a submerged shelf (360x50x80 grid cells). Note the ability to properly model the initial breaking (first three frames) and secondary splash up (last frame) phases.**

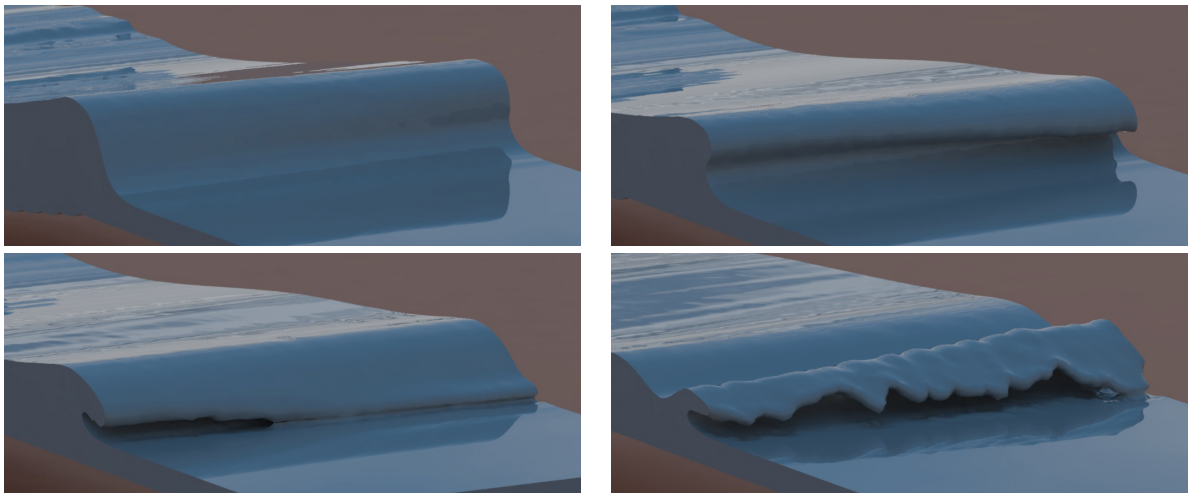


Figure 9: **Closeup of wave action seen in figure 8.**

Physically Based Modeling and Animation of Fire

Duc Quang Nguyen
Stanford University
Industrial Light & Magic
dqnguyen@stanford.edu

Ronald Fedkiw
Stanford University
Industrial Light & Magic
fedkiw@cs.stanford.edu

Henrik Wann Jensen
Stanford University
henrik@graphics.stanford.edu

Abstract

We present a physically based method for modeling and animating fire. Our method is suitable for both smooth (laminar) and turbulent flames, and it can be used to animate the burning of either solid or gas fuels. We use the incompressible Navier-Stokes equations to independently model both vaporized fuel and hot gaseous products. We develop a physically based model for the expansion that takes place when a vaporized fuel reacts to form hot gaseous products, and a related model for the similar expansion that takes place when a solid fuel is vaporized into a gaseous state. The hot gaseous products, smoke and soot rise under the influence of buoyancy and are rendered using a blackbody radiation model. We also model and render the blue core that results from radicals in the chemical reaction zone where fuel is converted into products. Our method allows the fire and smoke to interact with objects, and flammable objects can catch on fire.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Ray Tracing;

Keywords: flames, fire, smoke, chemical reaction, blackbody radiation, implicit surface, incompressible flow, stable fluids, vorticity confinement

1 Introduction

The modeling of natural phenomena such as fire and flames remains a challenging problem in computer graphics. Simulations of fluid behavior are in demand for special effects depicting smoke, water, fire and other natural phenomena. Fire effects are especially in demand due to the dangerous nature of this phenomenon. Fire simulations are also of interest for virtual reality effects, for example to help train fire fighters or to determine proper placement of exit signs in smoke filled rooms (i.e. so they can be seen). The interested reader is referred to [Rushmeier 1994].

Combustion processes can be loosely classified into two rather distinct types of phenomena: detonations and deflagrations. In both of these processes, chemical reactions convert fuel into hot gaseous products. Deflagrations are low speed events such as the fire and flames we address in this paper, while detonations are high speed events such as explosions where shock waves and other compressible effects are important, see e.g. [Yngve et al. 2000] and [Neff and Fiume 1999]. As low speed events, deflagrations can be modeled using the equations for incompressible flow (as opposed to those for compressible flow). Furthermore, since viscous effects are small, we use the incompressible inviscid Euler equations similar to [Fedkiw et al. 2001]. As noted therein, these equations can be solved efficiently using a semi-Lagrangian stable fluid approach, see e.g. [Staniforth and Cote 1991] and [Stam 1999].

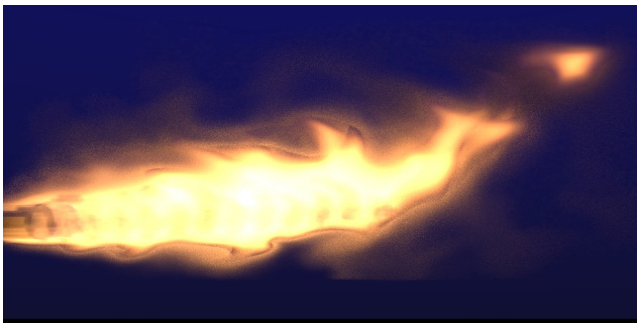


Figure 1: **A turbulent gas flame model of a flamethrower.**

An important, often neglected aspect of fire and flame modeling concerns the expansion of the fuel as it reacts to form hot gaseous products. This expansion is the reason for the visual fullness observed in many flames and is partly responsible for the visual turbulence as well. Since the incompressible equations do not account for expansion, we propose a simple thin flame model for capturing these effects. This is accomplished by using an implicit surface to represent the reaction zone where the gaseous fuel is converted into hot gaseous products. Although real reaction zones have a nonzero (but small) thickness, the thin flame approximation works well for visual modeling and has been used by scientists as well, see for example [Markstein 1964] who first proposed this methodology.

Our implementation of the thin flame model is as follows. First, a dynamic implicit surface is used to track the reaction zone where the gaseous fuel is converted into hot gaseous products. Then both the gaseous fuel and the hot gaseous products are separately modeled using independent sets of incompressible flow equations. Finally, these incompressible flow equations are updated together in a coupled fashion using the fact that both mass and momentum must be conserved as the gas reacts at the interface. While this gives rather pleasant looking laminar (smooth) flames, we include a vorticity confinement term, see [Steinhoff and Underhill 1994] and [Fedkiw et al. 2001], to model the larger scale turbulent flame structures that are difficult to capture on the relatively coarse grids used for efficiency reasons in computer graphics simulations. We also include other features important for visual simulation, such as the buoyancy effects generated by hot gases and the interaction of fire with flammable and nonflammable objects. We render the fire as a participating medium with blackbody radiation using a stochastic ray marching algorithm. In our rendering we pay careful attention

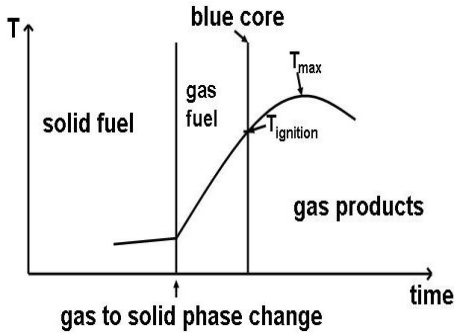


Figure 2: **Flame temperature profile for a solid (or gaseous) fuel.**

to the chromatic adaptation of the observer in order to get the correct colors of the fire.

2 Previous Work

A simple laminar flame was texture mapped onto a flame-like implicit primitive and then volume-traced by [Inakage 1989]. [Perry and Picard 1994] applied a velocity spread model from combustion science to propagate flames. [Chiba et al. 1994] computed the exchange of heat between objects by projecting the environment onto a plane. The spread of flame was a function of both the temperature and the concentration of fuel. [Stam and Fiume 1995] present a similar model in three spatial dimensions for the creation, extinguishing and spread of fire. The spread of the fire is controlled by the amount of fuel available, the geometry of the environment and the initial conditions. Their velocity field is predefined, and then the temperature and density fields are advected using an advection-diffusion type equation. They render the fire using a diffusion approximation which takes into account multiple scattering. [Bukowski and Sequin 1997] integrated the Berkeley Architectural Walkthrough Program with the National Institute of Standards and Technology's CFAST fire simulator. The integrated system creates a simulation based design environment for building fire safety systems.

Although we do not consider high-speed combustion phenomena such as detonations in this paper, there has been some notable work on this subject. [Musgrave 1997] concentrated on the explosive cloud portion of the explosion event using a fractal noise approach. [Neff and Fiume 1999] model and visualize the blast wave portion of an explosion based on a blast curve approach. [Mazarak et al. 1999] discuss the elementary blast wave equations, which were used to model exploding objects. They also show how to incorporate the blast wave model with a rigid body motion simulator to produce realistic animation of flying debris. Most recently, [Yngve et al. 2000] model the propagation of an explosion through the surrounding air using a computational fluid dynamics based approach to solve the equations for compressible, viscous flow. Their system includes two way coupling between solid objects and surrounding fluid, and uses the spectacular brittle fracture technology of [O'Brien and Hodgins 1999]. While the compressible flow equations are useful for modeling shock waves and other compressible phenomena, they introduce a very strict time step restriction associated with the acoustic waves. We use the incompressible flow equations instead to avoid this restriction making our method more computationally efficient.

3 Physically Based Model

We consider three distinct visual phenomena associated with flames. The first of these is the blue or bluish-green core seen in many flames. These colors are emission lines from intermediate chemical species, such as carbon radicals, produced during the chemical reaction. In the thin flame model, this thin blue core is located adjacent to the implicit surface. Therefore, in order to track this blue core, we need to track the movement of the implicit surface. The second visual phenomenon is the blackbody radiation emitted by the hot gaseous products, in particular the carbon soot. This is characterized by the yellowish-orange color we associate with fire. In order to model this with visual accuracy we need to track the temperatures associated with a flame as depicted in figure 2 (read from left to right). If the fuel is solid or liquid, the first step is the heating of the solid until it undergoes a phase change to the gaseous state. (Obviously, for gas fuels, we start in this gaseous state region in the figure.) Then the gas heats up until it reaches its ignition temperature corresponding to our implicit surface and the beginning of the thin blue core region. The temperature continues to increase as the reaction proceeds reaching a maximum before radiative cooling and mixing effects cause the temperature to decrease. As the temperature decreases, the blackbody radiation falls off until the yellowish-orange color is no longer visible. The third and final visual effect we address is the smoke or soot that is apparent in some flames after the temperature cools to the point where the blackbody radiation is no longer visible. We model this effect by carrying along a density variable in a fashion similar to the temperature. One could easily add particles to represent small pieces of soot, but our focus in this paper is the fire, not the smoke. For more details on smoke, see [Foster and Metaxas 1997], [Stam 1999] and [Fedkiw et al. 2001]. Figure 3 shows smoke coupled to our gas flame.

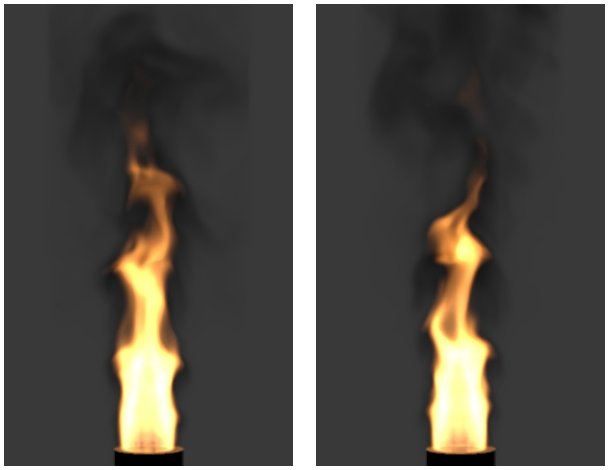


Figure 3: **The hot gaseous products and soot emit blackbody radiation that illuminates the smoke.**

3.1 Blue Core

Our implicit surface separates this gaseous fuel from the hot gaseous products and surrounding air. Consider for example the injection of gaseous fuel from a cylindrically shaped tube. If the fuel were not burning, then the implicit surface would simply move at the same velocity as the gaseous fuel being injected. However, when the fuel is reacting, the implicit surface moves at the velocity of the unreacted fuel plus a flame speed S that indicates how fast fuel is being converted into gaseous products. S indicates how fast the unreacted gaseous fuel is crossing over the implicit surface turning into hot gaseous products. The approximate surface area of the blue core, A_S, can be estimated with the following equation

$$v_f A_f = S A_S, \tag{1}$$

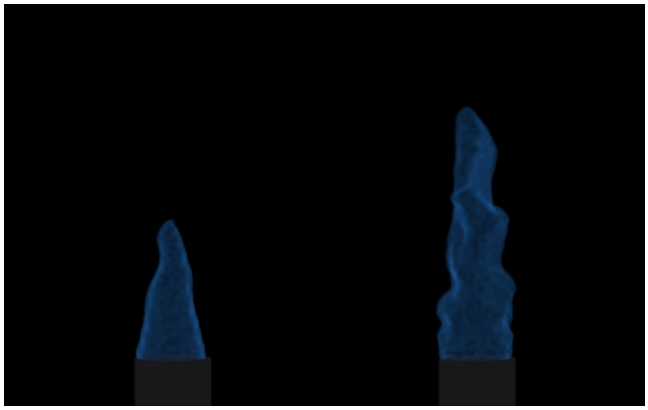


Figure 4: **Blue reaction zone cores for large (left) and small (right) values of the flame reaction speed S. Note the increased turbulence on the right.**

where v_f is the speed the fuel is injected across the injection surface with area A_f, e.g. A_f is the cross section of the cylindrical tube. This equation results from canceling out the density in the equation for conservation of mass. The left hand side is the fuel being injected into the region bounded by the implicit surface, and the right hand side is the fuel leaving this region crossing over the implicit surface as it turns into gaseous products. From this equation, we see that injecting more (less) gas is equivalent to increasing (decreasing) v_f resulting in a larger (smaller) blue core. Similarly, increasing (decreasing) the reaction speed S results in a smaller (larger) blue core. While we can turn the velocity up or down on our cylindrical jet, the reaction speed S is a property of the fuel. For example, S is approximately $.44m/s$ for a propane fuel that has been suitably premixed with oxidizer [Turns 1996]. (We use $S = .5m/s$ for most of our examples.) Figure 4 shows the effect of varying the parameter S. The smaller value of S gives a blue core with more surface area as shown in the figure.

This thin flame approximation is fairly accurate for premixed flames where the fuel and oxidizer are premixed so that the injected gas is ready for combustion. Non-premixed flames, commonly referred to as diffusion flames, behave somewhat differently. In a diffusion flame, the injected fuel has to mix with a surrounding oxidizer before it can combust. Figure 5 shows the injection of fuel out of a cylindrically shaped pipe. The cone shaped curve is the predicted location of the blue core for a premixed flame while the larger rounded curve is the predicted location of the blue core for a diffusion flame. As can be seen in the figure, diffusion flames tend to have larger cores since it takes a while for the injected fuel and surrounding oxidizer to mix. This small-scale molecular diffusion process is governed by a second order partial differential equation that is computationally costly model. Thus for visual purposes,

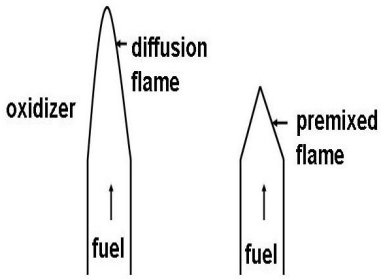


Figure 5: **Location of the blue reaction zone core for a premixed flame versus a diffusion (non-premixed) flame**

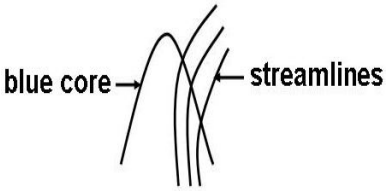


Figure 6: **Streamlines illustrating the path of individual fluid elements as they across the blue reaction zone core. The curved path is caused by the expansion of the gas as it reacts.**

we model diffusion flames with larger blue cores simply by using a smaller value of S than that used for a corresponding premixed flame.

3.2 Hot Gaseous Products

In order to get the proper visual look for our flames, it is important to track individual elements of the flow and follow them through their temperature histories given by figure 2. This is particularly difficult because the gas expands as it undergoes reaction from fuel to hot gaseous products. This expansion is important to model since it changes the trajectories of the gas and the subsequent look and feel of the flame as individual elements go through their temperature profile. Figure 3.2 shows some sample trajectories of individual elements as they cross over the reaction front. Note that individual elements do not go straight up as they pass through the reaction front, but instead turn outward due to the effects of expansion. It is difficult to obtain visually full turbulent flames without modeling this expansion effect. In fact, many practitioners resort to a number of low level hacks (and lots of random numbers) in an attempt to sculpt this behavior, while we obtain the behavior intrinsically by using the appropriate model. The expansion parameter is usually given as a ratio of densities, ρ_f/ρ_h where ρ_f is the density of the gaseous fuel and ρ_h is the density of the hot gaseous products. Figure 7 shows three flames side by side with increasing amounts of expansion from left to right. Note how increasing the expansion makes the flames appear fuller. We used $\rho_f = 1kg/m^3$ (about the density of air) for all three flames with $\rho_h = .2kg/m^3$, $.1kg/m^3$ and $.05kg/m^3$ from left to right.

We use one set of incompressible flow equations to model the fuel and a separate set of incompressible flow equations to model the hot gaseous products and surrounding airflow. We require a model for coupling these two sets of incompressible flow equations together across the interface in a manner that models the expansion that takes place across the reaction front. Given that mass and mo-

Figure 7: **Comparison of flame shapes for differing degrees of gaseous expansion. The amount of expansion increases from left to right making the flame appear fuller and more turbulent.**

mentum are conserved we can derive the following equations for the coupling across the thin flame front:

$$\rho_h(V_h - D) = \rho_f(V_f - D), \quad (2)$$

$$\rho_h(V_h - D)^2 + p_h = \rho_f(V_f - D)^2 + p_f, \quad (3)$$

where V_f and V_h are the normal velocities of the fuel and the hot gaseous products, and p_f and p_h are their pressures. Here, $D = V_f - S$ is the speed of the implicit surface in the normal direction. These equations indicate that both the velocity and the pressure are discontinuous across the flame front. Thus, we will need to exercise caution when taking derivatives of these quantities as is required when solving the incompressible flow equations. (Note that the tangential velocities are continuous across the flame front.)

3.3 Solid Fuels

When considering solid fuels, there are two expansions that need to be accounted for. Besides the expansion across the flame front, a similar expansion takes place when the solid is converted to a gas. However, S is usually relatively small for this reaction (most solids burn slowly in a visual sense), so we can use the boundary of the solid fuel as the reaction front. Since we do not currently model the pressure in solids, only equation 2 applies. We rewrite this equation as

$$\rho_f(V_f - D) = \rho_s(V_s - D), \quad (4)$$

where ρ_s and V_s are the density and the normal velocity of the solid fuel. Substituting $D = V_s - S$ and solving for V_f gives

$$V_f = V_s + \left(\rho_s/\rho_f - 1\right) S \quad (5)$$

indicating that the gasified solid fuel moves at the velocity of the solid fuel plus a correction that accounts for the expansion. We model this phase change by injecting gas out of the solid fuel at the appropriate velocity. This can be used to set arbitrary shaped solid objects on fire as long as they can be voxelized with a suitable surface normal assigned to each voxel indicating the direction of gaseous injection.

In figure 8, we simulate a campfire using two cylindrically shaped logs as solid fuel injecting gas out of the logs in a direction consistent with the local unit surface normal. Note the realistic rolling of the fire up from the base of the log. The ability to inject (or not inject) gaseous fuel out of individual voxels on the surface of a complex solid object allows us to animate objects catching on fire, burn different parts of an object at different rates or not at all (by using spatially varying injection velocities), and extinguish solid fuels simply by turning off the injection velocity. While building an animation system that allows the user to hand paint temporally and spatially varying injection velocities on the surface of solid objects is beyond the scope of this paper, it is the subject of future work.

4 Implementation

We use a uniform discretization of space into N^3 voxels with uniform spacing h. The implicit surface, temperature, density and pressure are defined at the voxel centers and are denoted $\phi_{i,j,k}$, $T_{i,j,k}$, $\rho_{i,j,k}$ and $p_{i,j,k}$ where $i,j,k = 1,\cdots,N$. The velocities are defined at the cell faces and we use half-way index notation: $u_{i+1/2,j,k}$ where $i = 0,\cdots,N$ and $j,k = 1,\cdots,N$; $v_{i,j+1/2,k}$ where $j = 0,\cdots,N$ and $i,k = 1,\cdots,N$; $w_{i,j,k+1/2}$ where $k = 0,\cdots,N$ and $i,j = 1,\cdots,N$.

Figure 8: **Two burning logs are placed on the ground and used to emit fuel. The crossways log on top is not lit so the flame is forced to flow around it.**

4.1 Level Set Equation

We track our reaction zone (blue core) using the level set method of [Osher and Sethian 1988] to track the moving implicit surface. We define ϕ to be positive in the region of space filled with fuel, negative elsewhere and zero at the reaction zone.

The implicit surface moves with velocity $\mathbf{w} = \mathbf{u}_f + S\mathbf{n}$ where $\mathbf{u}_f$ is the velocity of the gaseous fuel and the $S\mathbf{n}$ term governs the conversion of fuel into gaseous products. The local unit normal, $\mathbf{n} = \nabla\phi/|\nabla\phi|$ is defined at the center of each voxel using central differencing to approximate the necessary derivatives, e.g. $\phi_x \approx (\phi_{i+1,j,k} - \phi_{i-1,j,k})/2h$. Standard averaging of voxel face values is used to define $\mathbf{u}_f$ at the voxel centers, e.g. $u_{i,j,k} = (u_{i-1/2,j,k} + u_{i+1/2,j,k})/2$. The motion of the implicit surface is defined through

$$\phi_t = -\mathbf{w}\cdot\nabla\phi \quad (6)$$

and solved at each grid point using

$$\phi^{new} = \phi^{old} - \Delta t\left(w_1\phi_x + w_2\phi_y + w_3\phi_z\right) \quad (7)$$

and an upwind differencing approach to estimate the spatial derivatives. For example, if $w_1 > 0$, $\phi_x \approx (\phi_{i,j,k} - \phi_{i-1,j,k})/h$. Otherwise if $w_1 < 0$, $\phi_x \approx (\phi_{i+1,j,k} - \phi_{i,j,k})/h$. This simple approach is efficient and produces visually appealing blue cores.

To keep the implicit surface well conditioned, we occasionally adjust the values of ϕ in order to keep ϕ a signed distance function with $|\nabla\phi| = 1$. First, interpolation is used to reset the values of ϕ at voxels adjacent to the $\phi = 0$ isocontour (which we don't want to

move since it is the visual location of the blue core). Then we march out from the zero isocontour adjusting the values of ϕ at the other grid points as we cross them. [Tsitsiklis 1995] showed that this could be accomplished in an accurate, optimal and efficient manner solving quadratic equations and sorting points with a binary heap data structure. Later, [Sethian 1996] proposed the finite difference formulation of this algorithm that we currently use.

4.2 Incompressible Flow

We model the flow of the gaseous fuel and the hot gaseous products using a separate set of incompressible Euler equations for each. Incompressibility is enforced through conservation of mass (or volume), i.e. $\nabla \cdot \mathbf{u} = 0$ where $\mathbf{u} = (u, v, w)$ is the velocity field. The equations for the velocity

$$\mathbf{u}_t = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla p/\rho + \mathbf{f} \tag{8}$$

are solved for in two parts. First, we use this equation to compute an intermediate velocity $\mathbf{u}^*$ ignoring the pressure term, and then we add the pressure (correction) term using

$$\mathbf{u} = \mathbf{u}^* - \Delta t \nabla p/\rho. \tag{9}$$

The key idea to this splitting method is illustrated by taking the divergence of equation 9 to obtain

$$\nabla \cdot \mathbf{u} = \nabla \cdot \mathbf{u}^* - \Delta t \nabla \cdot (\nabla p/\rho) \tag{10}$$

and then realizing that we want $\nabla \cdot \mathbf{u} = 0$ to enforce mass conservation. Thus the left hand side of equation 10 should vanish leaving a Poisson equation of the form

$$\nabla \cdot (\nabla p/\rho) = \nabla \cdot \mathbf{u}^*/\Delta t \tag{11}$$

that can be solved to find the pressure needed for updating equation 9.

We use a semi-Lagrangian stable fluids approach for finding the intermediate velocity $\mathbf{u}^*$ and refer the reader to [Stam 1999] and [Fedkiw et al. 2001] for the details. Since we use two sets of incompressible flow equations, we need to address the stable fluid update when a characteristic traced back from one set of incompressible flow equations crosses the implicit surface and queries the velocities from the other set of incompressible flow equations. Since the normal velocity is discontinuous across the interface, the straightforward stable fluids approach fails to work. Instead, we need to use the balance equation 2 for conservation of mass to correctly interpolate a velocity.

Suppose we are solving for the hot gaseous products and we interpolate across the interface into a region where a velocity from the gaseous fuel might incorrectly be used. Instead of using this value, we compute a ghost value as follows. First, we compute the normal velocity of the fuel, $V_f = \mathbf{u}_f \cdot \mathbf{n}$. Then we use the balance equation 2 to find a ghost value for V_h^G as

$$V_h^G = V_f + (\rho_f/\rho_h - 1)\, S. \tag{12}$$

Since the tangential velocities are continuous across the implicit surface, we combine this new normal velocity with the existing tangential velocity to obtain

$$\mathbf{u}_h^G = V_h^G \mathbf{n} + \mathbf{u}_f - (\mathbf{u}_f \cdot \mathbf{n})\mathbf{n} \tag{13}$$

as a ghost value for the velocity of the hot gaseous products in the region where only the fuel is defined. This ghost velocity can then be used to correctly carry out the stable fluids update. Since both $\mathbf{n}$ and $\mathbf{u}_f$ are defined throughout the region occupied by the fuel, and ρ_f, ρ_h and S are known constants, a ghost cell value for the hot gaseous products, $\mathbf{u}_h^G$, can be found anywhere in the fuel region (even quite far from the interface) by simply algebraically evaluating the right hand side of equation 13. [Nguyen et al. 2001] showed that this ghost fluid method, invented in [Fedkiw et al. 1999], could be used to compute physically accurate engineering simulations of deflagrations.

After computing the intermediate velocity $\mathbf{u}^*$ for both sets of incompressible flow equations, we solve equation 11 for the pressure and finally use equation 9 to find our new velocity field. Equation 11 is solved by assembling and solving a linear system of equations for the pressure as discussed in more detail in [Foster and Fedkiw 2001] and [Fedkiw et al. 2001]. Once again, we need to exercise caution here since the pressure is discontinuous across the interface. Using the ghost fluid method and equation 3, we can obtain and solve a slightly modified linear system incorporating this jump in pressure. We refer the reader to [Nguyen et al. 2001] for explicit details and a demonstration of the physical accuracy of this approach in the context of deflagration waves.

The temperature affects the fluid velocity as hot gases tend to rise due to buoyancy. We use a simple model to account for these effects by defining external forces that are directly proportional to the temperature

$$\mathbf{f}_{buoy} = \alpha (T - T_{air})\, \mathbf{z}, \tag{14}$$

where $\mathbf{z} = (0, 0, 1)$ points in the upward vertical direction, T_{air} is the ambient temperature of the air and α is positive constant with the appropriate units.

Fire, smoke and air mixtures contain velocity fields with large spatial deviations accompanied by a significant amount of rotational and turbulent structure on a variety of scales. Nonphysical numerical dissipation damps out these interesting flow features, so we aim to add them back on the coarse grid. We use the vorticity confinement technique invented by Steinhoff (see e.g. [Steinhoff and Underhill 1994]) and used by [Fedkiw et al. 2001] to generate the swirling effects for smoke. The first step in generating the small scale detail is to identify the vorticity $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ as the source of this small scale structure. Each small piece of vorticity can be thought of as a paddle wheel trying to spin the flow field in a particular direction. Normalized vorticity location vectors, $\mathbf{N} = \nabla|\boldsymbol{\omega}|/|\nabla|\boldsymbol{\omega}||$ simply point from lower concentrations of vorticity to higher concentrations. Using these, the magnitude and direction of the vorticity confinement (paddle wheel) force is computed as

$$\mathbf{f}_{conf} = \varepsilon h (\mathbf{N} \times \boldsymbol{\omega}), \tag{15}$$

where $\varepsilon > 0$ and is used to control the amount of small scale detail added back into the flow field. The dependence on h guarantees that as the mesh is refined the physically correct solution is still obtained. All these quantities can be evaluated in a straightforward fashion as outlined in [Fedkiw et al. 2001].

Usually a standard CFL time step restriction dictates that the time step $\triangle t$ should be limited by $\triangle t < h/|\mathbf{u}|_{max}$ where $|\mathbf{u}|_{max}$ is the maximum velocity in the flow field. While this is true for our level set equation 6 with $\mathbf{u}$ replaced by $\mathbf{w}$, the combination of the semi-Lagrangian discretization and the ghost fluid method allows us to take a much larger time step for the incompressible flow equations. We choose our incompressible flow time step to be about five times bigger than that dictated by applying the CFL condition to the level set equation, and then stably update ϕ using substeps. This reduces the number of times one needs to solve for the pressure, which is the most expensive part of the calculation, by a factor of five.

4.3 Temperature and Density

The temperature profile has great effect on how we visually perceive flames, and we need to generate a temperature time history for fluid elements that behaves as shown in figure 2. Since this figure depicts a time history of the temperature of fluid elements, we

need a way to track individual fluid elements as they cross over the blue core and rise upward due to buoyancy. In particular, we need to know how much time has elapsed since a fluid element has passed through the blue core so that we can assign an appropriate temperature to it. This is easily accomplished using a reaction coordinate variable Y governed by the equation

$$Y_t = -(\mathbf{u}\cdot\nabla)Y - k, \tag{16}$$

where k is a positive constant which we take to be 1 (larger or smaller values can be used to get a good numerical variation of Y in the flame). Ignoring the convection term, $Y_t = -1$ can be solved exactly to obtain $Y(t) = -t + Y(0)$. If we set $Y(0) = 1$ in the region of space occupied by the gaseous fuel and solve equation 16 for Y, then the local value of $1-Y$ is equal to the total time elapsed since a fluid element crossed over the blue reaction core.

We solve equation 16 using the semi-Lagrangian stable fluids method to first update the convection term obtaining an intermediate value Y^*. Then we separately integrate the source term analytically so it too is stable for large time steps, i.e. $Y^{new} = -k\Delta t + Y^*$.

We can now use the values of Y to assign temperature values to the flow. Since $T_{ignition}$ is usually below the visual blackbody emission threshold, the temperature we set inside the blue core is usually not important. Therefore, we can set $T = T_{ignition}$ for the points inside the blue core. The region between the blue core and the maximum temperature in figure 2 is important since it models the rise in temperature due to the progress of a complex chemical reaction (which we do not model for the sake of efficiency). Here the animator has a lot of freedom to sculpt temperature rise curves and adjust how the mapping corresponds to the local Y values. For example, one could use $T = T_{ignition}$ at $Y = 1$, $T = T_{max}$ at $Y = .9$ and use a linear temperature function for the in between values of $Y \in (.9, 1)$. For large flames, this temperature rise interval will be compressed too close to the blue core for our grid to resolve. In these instances we use the ghost fluid method to set $T = T_{max}$ for any characteristic that looks across the blue core into the gaseous fuel region. The blue core then "spits" out gas at the maximum temperature that immediately starts to cool off, i.e. there is no temperature rise region. In fact, we did not find it necessary to use the temperature rise region in our examples as we are interested in larger scale flames, but this temperature rise region would be useful, for example, when modeling candle.

The animator can also sculpt the temperature falloff region to the right of figure 2. However, there is a physically correct, viable (i.e. computationally cheap) alternative. For the values of Y in the temperature falloff region, we simply solve

$$T_t = -(\mathbf{u}\cdot\nabla)T - c_T\left(\frac{T - T_{air}}{T_{max} - T_{air}}\right)^4 \tag{17}$$

which is derived from conservation of energy. Similar to equation 16, we solve this equation by first using the semi-Lagrangian stable fluids method to solve for the convection term. Then we integrate the fourth power term analytically to cool down the flame at a rate governed by the cooling constant c_T.

Similar to the temperature curve in figure 2, the animator can sculpt a density curve for smoke and soot formation. The density should start low and increase as the reaction proceeds. In the temperature falloff region, the animator can switch from the density curve to a physically correct equation

$$\rho_t = -(\mathbf{u}\cdot\nabla)\rho \tag{18}$$

that can (once again) be solved using the semi-Lagrangian stable fluids method. Again, we did not find it necessary to sculpt densities for our particular examples.

5 Rendering of Fire

Fire is a participating medium. It is more complex than the types of participating media (e.g. smoke and fog) that are typically encountered in computer graphics since fire emits light. The region that creates the light-energy typically has a complex shape, which makes it difficult to sample. Another complication with fire is that the fire is bright enough that our eyes adapt to its color. This chromatic adaptation is important to account for when displaying fire on a monitor. In this section, we will first describe how we simulate the scattering of light within a fire-medium. Then, we will detail how to properly integrate the spectral distribution of power in the fire and account for chromatic adaptation.

5.1 Light Scattering in a Fire Medium

Fire is a blackbody radiator and a participating medium. The properties of a participating medium are described by the scattering, absorption and emission properties. Specifically, we have the scattering coefficient, σ_s, the absorption coefficient, σ_a, and the extinction coefficient, $\sigma_t = \sigma_a + \sigma_s$. These coefficients specify the amount of scattering, absorption and extinction per unit-distance for a beam of light moving through the medium. The spherical distribution of the scattered light at a location is specified by a phase-function, p. We use the Henyey-Greenstein phase-function [Henyey and Greenstein 1941]

$$p(\vec{\omega}\cdot\vec{\omega}') = \frac{1-g^2}{4\pi(1+g^2-2g\vec{\omega}\cdot\vec{\omega}')^{1.5}}. \tag{19}$$

Here, $g \in [-1, 1]$ is the scattering anisotropy of the medium, $g > 0$ is forward scattering, $g < 0$ is backward scattering, while $g = 0$ is isotropic scattering. Note that the distribution of the scattered light only depends on the angle between the incoming direction, $\vec{\omega}$, and the outgoing direction, $\vec{\omega}'$.

Light transport in participating media is described by an integro-differential equation, the radiative transport equation [Siegel and Howell 1981]:

$$\begin{aligned}(\vec{\omega}\cdot\nabla)L_\lambda(x,\vec{\omega}) \quad = \quad & -\sigma_t(x)L_\lambda(x,\vec{\omega}) + \\ & \sigma_s(x)\int_{4\pi} p(\vec{\omega},\vec{\omega}')L_\lambda(x,\vec{\omega}')d\vec{\omega}' + \\ & \sigma_a(x)L_{e,\lambda}(x,\vec{\omega}).\end{aligned} \tag{20}$$

Here, L_λ is the spectral radiance, and $L_{e,\lambda}$ is the emitted spectral radiance. Note that σ_s, σ_a, and σ_t vary throughout the medium and therefore depend on the position x.

We solve Equation 20 to estimate the radiance distribution in the medium by using a stochastic adaptive ray marching algorithm which recursively samples multiple scattering. In highly scattering media this approach is costly; however, we are concerned about fire which is a blackbody radiator (no scattering, only absorption) that creates a low-albedo smoke (the only scattering part of the fire-medium). This makes the Monte Carlo ray tracing approach practical.

To estimate the radiance along a ray traversing the medium, we split the ray into short segments. For a given segment, n, the scattering properties of the medium are assumed constant, and the radiance, L_n, at the start of the segment is computed as:

$$\begin{aligned}L_{n,\lambda}(x,\vec{\omega}) \quad = \quad & e^{-\sigma_t\Delta x}L_{(n-1),\lambda}(x+\Delta x,\vec{\omega}) + \\ & L_\lambda(x,\vec{\omega}')p(\vec{\omega}\cdot\vec{\omega}')\sigma_s\Delta x + \\ & \sigma_a L_{e,\lambda}(x)\Delta x.\end{aligned} \tag{21}$$

This equation is evaluated recursively to compute the total radiance at the origin of the ray. Δx is the length of the segment, L_{n-1} is the radiance at the beginning of the next segment, and $\vec{\omega}'$ is a sample

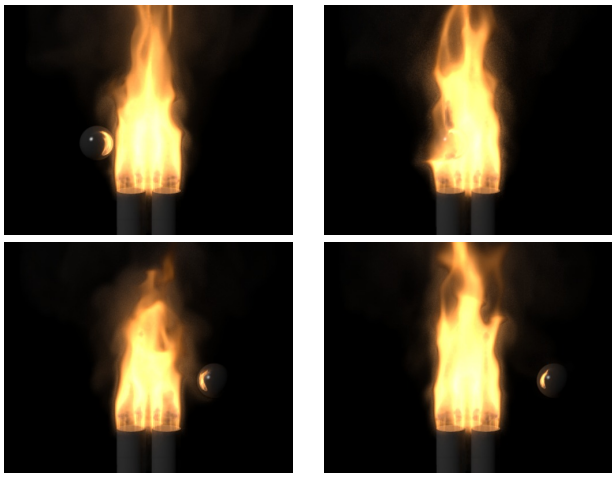


Figure 9: **A metal ball passes through and interacts with a gas flame.**

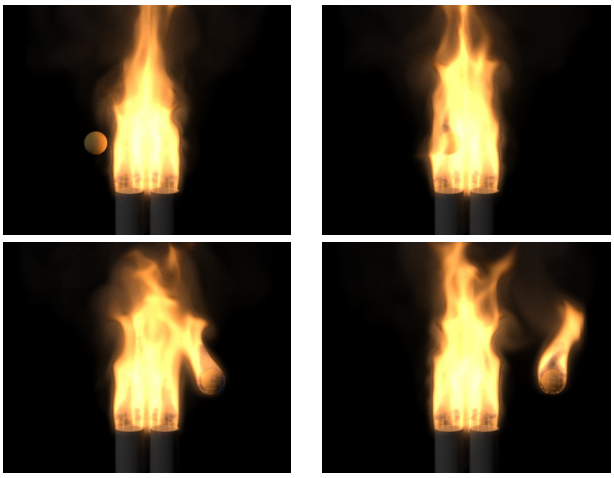


Figure 10: **A flammable ball passes through a gas flame and catches on fire.**

direction for a new ray that evaluates the indirect illumination in a given direction for the segment. We find the sample direction by importance sampling the Henyey-Greenstein phase function. Note that we do not explicitly sample the fire volume; instead we rely on the Monte Carlo sampling to pick up energy as sample rays hit the fire. This strategy is reasonably efficient in the presence of the low-albedo smoke generated by the fire.

The emitted radiance is normally ignored in graphics, but for fire it is an essential component. For a blackbody we can compute the emitted spectral radiance using Planck's formula:

$$L_{e,\lambda}(x) = \frac{2C_1}{\lambda^5(e^{C_2/(\lambda T)} - 1)}, \tag{22}$$

where T is the temperature, $C_1 \approx 3.7418 \cdot 10^{-16} W m^2$, and $C_2 \approx 1.4388 \cdot 10^{-2} m^{\circ} K$ [Siegel and Howell 1981]. In the next section, we will describe how to properly render fire taking this spectral distribution of emitted radiance into account.

5.2 Reproducing the Color of Fire

Accurately reproducing the colors of fire is critical for a realistic fire rendering. The full spectral distribution can be obtained directly by using Planck's formula for spectral radiance when performing the ray marching. This spectrum can then be converted to RGB before being displayed on a monitor. To get the right colors of fire out of this process it is necessary to take into account the fact that our eyes adapt to the spectrum of the fire.

To compute the chromatic adaptation for fire, we use a von Kries transformation [Fairchild 1998]. We assume that the eye is adapted to the color of the spectrum for the maximum temperature present in the fire. We map the spectrum of this white point to the LMS cone responsivities (L_w, M_w, S_w). This enables us to map a spectrum to the monitor as follows. We first integrate the spectrum to find the raw XYZ tristimulus values (X_r, Y_r, Z_r). We then find the adapted XYZ tristimulus values (X_a, Y_a, Z_a) as:

$$\begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix} = \mathbf{M}^{-1} \begin{bmatrix} 1/L_w & 0 & 0 \\ 0 & 1/M_w & 0 \\ 0 & 0 & 1/S_w \end{bmatrix} \mathbf{M} \begin{bmatrix} X_r \\ Y_r \\ Z_r \end{bmatrix}. \tag{23}$$

Here, $\mathbf{M}$ maps the XYZ colors to LMS (consult [Fairchild 1998] for the details). Finally, we map the adapted XYZ tristimulus values to the monitor RGB space using the monitor white point.

In our implementation, we integrate the spectrum of the blackbody at the source (e.g. when emitted radiance is computed); we then map this spectrum to RGB before using it in the ray marcher. This is much faster than doing a full spectral participating media simulation, and we found that it is sufficiently accurate, since we already assume that the fire is the dominating light source in the scene when doing the von Kries transformation.

6 Results

Figure 1 shows a frame from a simulation of a flamethrower. We used a domain that was 8 meters long with 160 grid cells in the horizontal direction ($h = .05$). The flame was injected at $30m/s$ out of a cylindrical pipe with diameter $.4m$. We used $S = .5m/s$, $rho_f = 1kg/m^3$, $rho_h = .01kg/m^3$, $c_T = 3000K/s$ and $\alpha = .15m/(Ks^2)$. The vorticity confinement parameter was set to $\varepsilon = 16$ for the gaseous fuel and to $\varepsilon = 60$ for the hot gaseous products. The simulation cost was approximately 3 minutes per frame using a Pentium IV.

Solid objects are treated by first tagging all the voxels inside the object as occupied. Then all the occupied voxel cell faces have their velocity set to that of the object. The temperature at the center of occupied voxels is set to the object's temperature and the (smoke) density is set to zero. Figure 9 shows a metal sphere as it passes through and interacts with a gas fire. Note the reflection of the fire on the surface of the sphere. For more details on object interactions with liquids and gases see [Foster and Fedkiw 2001] and [Fedkiw et al. 2001].

Since we have high temperatures (i.e. fire) in our flow field, we allow our objects to heat up if their temperature is lower than that of their surroundings. We use a simple conduction model where we increase the local temperature of an object depending on the surrounding air temperature and object temperature as well as the time step Δt. Normally, the value of the implicit surface is set to a negative value of h at the center of all voxels occupied by objects indicating that there is no available fuel. However, we can easily model ignition for objects we designate as flammable. Once the temperature of a voxel inside an object increases above a predefined threshold indicating ignition, we change the value of the implicit surface in that voxel from $-h$ to h indicating that it contains fuel. In addition, those voxel's faces have their velocities augmented above the object velocity by an increment in the direction normal to the object surface indicating that gaseous fuel is being injected according to the phase change addressed earlier for solid fuels. In figure 10, we illustrate this technique with a spherical ball that heats up and subsequently catches on fire as it passes through the flame. Both this flammable ball and the metal ball were com-

puted on a $120 \times 120 \times 120$ grid at approximately 5 minutes per frame.

7 Conclusion

We have presented a physically based model for animating and rendering fire and flames. We demonstrated that this model could be used to produce realistic looking turbulent flames from both solid and gaseous fuels. We showed plausible interaction of our fire and smoke with objects, including ignition of objects by the flames.

8 Acknowledgment

Research supported in part by an ONR YIP and PECASE award N00014-01-1-0620, NSF DMS-0106694, NSF ACI-0121288, and the DOE ASCI Academic Strategic Alliances Program (LLNL contract B341491).

References

BUKOWSKI, R., AND SEQUIN, C. 1997. Interactive Simulation of Fire in Virtual Building Environments. In *Proceedings of SIGGRAPH 1997*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 35–44.

CHIBA, N., MURAOKA, K., TAKAHASHI, H., AND MIURA, M. 1994. Two dimensional Visual Simulation of Flames, Smoke and the Spread of Fire. *The Journal of Visualization and Computer Animation 5*, 37–53.

FAIRCHILD, M. 1998. *Color Appearance Models*. Addison Wesley Longman, Inc.

FEDKIW, R., ASLAM, T., MERRIMAN, B., AND OSHER, S. 1999. A Non-oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (The Ghost Fluid Method). *J. Comput. Phys. 152*, 457.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual Simulation of Smoke. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 15–22.

FOSTER, N., AND FEDKIW, R. 2001. Practical Animation of Liquids. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 23–30.

FOSTER, N., AND METAXAS, D. 1997. Modeling the Motion of a Hot, Turbulent Gas. In *Proceedings of SIGGRAPH 97*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 181–188.

HENYEY, L., AND GREENSTEIN, J. 1941. Diffuse radiation in the galaxy. *Astrophysics Journal 93*, 70–83.

INAKAGE, M. 1989. A Simple Model of Flames. In *Proceedings of Computer Graphics International 89*, Springer-Verlag, 71–81.

MARKSTEIN, G. H. 1964. *Nonsteady Flame Propagation*. Pergamon, Oxford.

MAZARAK, O., MARTINS, C., AND AMANATIDES, J. 1999. Animating Exploding Objects. In *Proceedings of Graphics Interface 99*, 211–218.

MUSGRAVE, F. K. 1997. Great Balls of Fire. In *SIGGRAPH 97 Animation Sketches, Visual Procedings*, ACM SIGGRAPH, 259–268.

NEFF, M., AND FIUME, E. 1999. A Visual Model for Blast Waves and Fracture. In *Proceedings of Graphics Interface 99*, 193–202.

NGUYEN, D., FEDKIW, R., AND KANG, M. 2001. A Boundary Condition Capturing Method for Incompressible Flame Discontinuities. *J. Comput. Phys. 172*, 71–98.

O'BRIEN, J. F., AND HODGINS, J. K. 1999. Graphical Modeling and Animation of Brittle Fracture. In *Proceedings of SIGGRAPH 1999*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 137–146.

OSHER, S., AND SETHIAN, J. A. 1988. Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formualtions. *J. Comput. Phys. 79*, 12.

PERRY, C., AND PICARD, R. 1994. Synthesizing Flames and their Spread. *SIGGRAPH 94 Technical Sketches Notes* (July).

RUSHMEIER, H. 1994. Rendering Participating Media: Problems and Solutions from Application Areas. In *Proceedings of the 5th Eurographics Workshop on Rendering*, 35–56.

SETHIAN, J. 1996. A Fast Marching Level Set Method for Monotonically Advancing Fronts. *Proc. Nat. Acad. Sci. 93*, 1591–1595.

SIEGEL, R., AND HOWELL, J. 1981. *Thermal Radiation Heat Transfer*. Hemisphere Publishing Corp., Washington, DC.

STAM, J., AND FIUME, E. 1995. Depicting Fire and Other Gaseous Phenomena Using Diffusion Process. In *Proceedings of SIGGRAPH 95*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 129–136.

STAM, J. 1999. Stable Fluids. In *SIGGRAPH 99 Conference Proceedings, Annual Conference Series*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 121–128.

STANIFORTH, A., AND COTE, J. 1991. Semi-Lagrangian Integration Schemes for Atmospheric Models: A Review. *Monthly Weather Review 119*, 2206–2223.

STEINHOFF, J., AND UNDERHILL, D. 1994. Modification of the Euler Equations for "Vorticity Confinement": Application to the Computation of Interacting Vortex Rings. *Physics of Fluids 6*, 8, 2738–2744.

TSITSIKLIS, J. 1995. Efficient Algorithms for Globally Optimal Trajectories. *IEEE Transactions on Automatic Control 40*, 1528–1538.

TURNS, S. R. 1996. *An Introduction to Combustion*. McGraw-Hill, Inc.

YNGVE, G. D., O'BRIEN, J. F., AND HODGINS, J. K. 2000. Animating Explosions. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 29–36.