Object-Based Image Editing

William A. Barrett barrett@cs.byu.edu
Alan S. Cheney cheneya@cs.byu.edu
Department of Computer Science, Brigham Young University

Figure 1 Animation of a static, digital photo: Mrs. Potato [Hasbro 2001] Takes a Bow. (a) Digital photo of a toy on a carpet background. Frames c–e are edited versions of *this* digital photo – *not* computer graphic models. (b) Image objects (arms, eyes, nose, mouth, ear, tongue, body) are selected in about 27 seconds (~3 sec./object) (c) Object editing: arms are stretched and bent in real time using gesture motions with a mouse; ears, eyes, nose mouth and tongue (top of head) are shifted down, scaled and warped to simulate foreshortening. Background texture is filled in automatically. (d) Object editing continues. (e) Final pose: Foreshortened body occludes shoes; arm occludes body; ears occlude arms; "texture painting" was applied to fix imperfect background filling above eyes. All editing operations were performed interactively in a total of about 2 minutes.

Abstract

We introduce Object-Based Image Editing (OBIE) for real-time animation and manipulation of static digital photographs. Individual image objects (such as an arm or nose, Figure 1) are selected, scaled, stretched, bent, warped or even deleted (with automatic *hole filling*) - *at the object, rather than the pixel level* - using simple gesture motions with a mouse. OBIE gives the user direct, local control over object shape, size, and placement while dramatically reducing the time required to perform image editing tasks.

Object selection is performed by manually collecting (subobject) regions detected by a watershed algorithm. Objects are tessellated into a triangular mesh, allowing shape modification to be performed in real time using OpenGL's texture mapping hardware.

Through the use of *anchor points*, the user is able to interactively perform editing operations on a whole object, or just part(s) of an object - including moving, scaling, rotating, stretching, bending, and deleting. *Indirect* manipulation of object shape is also provided through the use of sliders and Bezier curves. Holes created by movement are filled in real-time based on surrounding texture.

When objects stretch or scale, we provide a method for preserving *texture granularity* or scale. We also present a *texture brush*, which allows the user to "paint" texture into different parts of an image, using existing image texture(s).

OBIE allows the user to perform interactive, high-level editing of image objects in a few seconds to a few ten's of seconds

Keywords: Image Editing, Image-based rendering, Animation, Texture Synthesis, Image Warping

1 INTRODUCTION

Object-based editing operations have traditionally been limited to well defined graphical objects (circles, rectangles, etc.) created in a drawing or modeling application. In contrast, image editing programs, such as Photoshop provide a rich assortment of pixel-based editing tools (cloning, pixel brushing, etc.) but limit object-based editing operations, such as scaling, warping, rotation or recoloring, to global manipulation of a bounding box over groups of selected pixels.

Object-based image editing (OBIE), presented in this paper, allows the user to animate static objects in digital photographs with direct, local object (and subobject) control. Tools are created for object selection, triangulation, stretching and bending, with texture preservation and background filling, producing new gestures and poses with foreshortening and self-occlusion - all interactively. This greatly increases the freedom with which image objects can be edited and animated, while dramatically reducing the time needed to perform such editing operations. OBIE tools make a fundamental contribution to the problem of image editing by changing the granularity of editing operations from the pixel to the object (or subobject) level. These tools operate in a way that is more natural to the object's topography, rather than requiring that editing operations be limited to a standard rectangular grid. OBIE can also be used to animate objects in static digital frames, as shown in Figures 1, 16, 20 and 21.

There are three image-editing methods currently in use, each of which have their own drawbacks. Pixel-based methods (clone tools, pixel painting, and nudge warping [Adobe 2000; GIMP; Scansoft 2001]) push pixels around to produce surprisingly good results, but are very time-intensive and do not allow direct, object-level manipulation. Region-of-interest (ROI) methods such as rectangle-based tools [Adobe 2000; GIMP] limit pixel modification to global manipulation of an axis-aligned bounding box and do not update the pixel colors in the region until the mouse movement stops. One ROI method [Elder and Goldberg 1998] *does* allow some object-level control, but requires the user to perform lengthy contour grouping operations and to work with disconnected contours vs. well-defined regions. Image-based editing methods (such as warping with thin-plate splines and with radial basis functions [Beier and Neely 1992; Bookstein 1989; ScanSoft 2001]) affect the entire image, like a rubber sheet, but do not allow for efficient, local control of an object's shape independent of surrounding background.

Recent work in segmentation and semi-automated object selection [McInerney and Terzopoulos 2000; Mortensen and Barrett 2001; Mortensen and Barrett 1995; Mortensen and Barrett 1998; Mortensen and Barrett 1999; Mortensen et al. 2000; Reese 1999] as well as texture synthesis [Efros and Leung 1999; Efros and Freeman 2001; Harrison 2001; Praun et al. 2000; Wei and Levoy 2000; Xu et al. 2000] has accelerated the interest in object-based image editing. In this paper we integrate into a single, interactive framework, a suite of tools for object selection and editing (with automatic hole-filling),

that provides real-time feedback while preserving the scale and variation native to object and background textures. Objects can be manipulated directly with a mouse or indirectly using *curve deformers* to attenuate length, thickness, and rotational bend. We also introduce texture painting for advanced cloning operations and creation of a variety of painterly effects.

2 OBJECT SELECTION

Before performing editing operations, the object is selected and broken down into a triangular network that captures the subobject detail. Object selection is based on *tobogganing*, a watershed algorithm [Vincent and Soille 1991] used in recently published interactive segmentation techniques [Mortensen and Barrett 1999; Mortensen et al. 2000; Reese 1999]. Watershed tobogganing amounts to sliding downward in the gradient magnitude image, to points of lowest gradient magnitude (base points). Pixels which slide to the same base point define *catchment basins* (~5-15 pixels). An algorithm for automated grouping of catchment basins, based on statistical similarity, was first developed for Intelligent Paint object selection [Reese 1999], and subsequently applied to Intelligent Scissors [Mortensen and Barrett 1999]. Grouped catchment basins consist of a few ten's to hundred's of pixels, referred to in this paper as *TRAP*s (**T**obogganed **R**egions of **A**ccumulated **P**lateau**s**). While TRAPs still produce an oversegmented image (Figure 2a), they adhere well to object and subobject edges and correspond nicely to the level of subobject detail needed without grouping object and background TRAPs.

We begin by computing TRAPs for the entire image (Figure 2a). All image pixels are associated with their corresponding TRAPs and labelled accordingly for later tagging and object definition. Note that in the zoomed portion of the image (Figure 2b), although the object is oversegmented, the TRAPs do a reasonable job of *automatically* capturing the subobject detail (fingers, hand, wrist).

Object selection is performed by manually collecting the TRAPs belonging to the object: when the user clicks the mouse on the image, the TRAP region containing that pixel is selected, adding it to the selected object and storing it in the selection list. Figure 2c shows the hand TRAPs comprising the object selected in blue.

Multiple TRAPs can be selected together by dragging a selection box over the object of interest. Deselection can be similarly accomplished by clicking on a TRAP already selected, which toggles it off, and removes it from the selection list. In the case that an object will move or shrink, creating a hole, something needs to be done to fill those exposed pixels. As will be explained later, surrounding (or "background") TRAPs are used in filling in these holes. When an object is selected, participating background TRAPs can be automatically selected, using a breadth-first connected component algorithm - dealing with TRAPs instead of pixels as the base components - continuing outward for as many layers out as requested. The new layers of TRAPs form a background object.

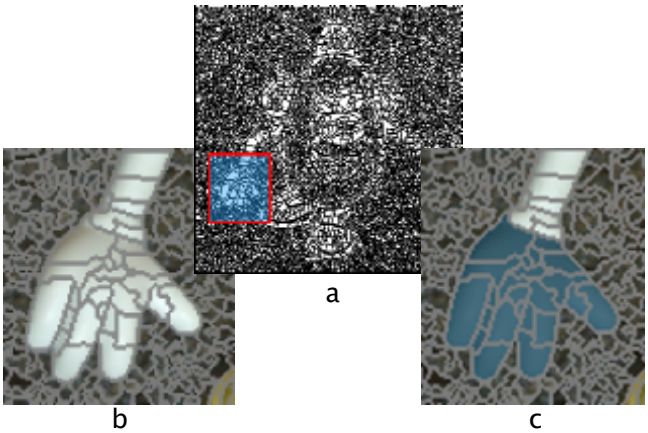


Figure 2: TRAP Tagging. (a) TRAP boundaries calculated in about 4 seconds for a 512^2 image. (b) Close-up of hand with fine-grained TRAP boundaries overlaid. (c) Tagged TRAPs shown in blue.

3 OBJECT REPRESENTATION

The object boundary is detected by applying a contour following algorithm to the union of the selected TRAPs, treated as a single binary object.

In order to manipulate the object in real-time with OpenGL we create a Delaunay triangular network from the object boundary and its associated TRAPs. Triangulation of the object requires triangular vertices (nodes) along the object boundary as well as nodes internal to the object. Boundary nodes are determined by polygonalizing the boundary with a simple recursive divide & conquer algorithm. The vertices of the resulting polygon define the boundary nodes. The number of boundary nodes is determined by specifying the tolerance level of the polygonal line fit.

Internal nodes (acting as Steiner points [Bern and Eppstein 1992] for the triangulation) are placed at base points (valleys) and where three or more TRAP boundaries (ridges) meet to form junctions (Figure 3) in the gradient magnitude image. A Delaunay triangulation is then performed on the object, using the specified internal nodes and an algorithm from [Shewchuk 1996], resulting in the creation of more optimally-shaped triangles. If a true Delaunay triangulation is not possible, because there are not enough input nodes given, then a close approximation to it is found. Alternative, but more computational triangulation schemes, such as found in [Yu et. al 2001] are also possible.

Triangular representation of individual TRAPs occurs in the same way by treating TRAPs as objects, in which case the only internal node is the base point. Triangular representation of individual TRAPs is necessary for the implementation of texture painting (discussed later) where TRAP connectivity is not required. Figure 4 shows the triangulation of individual hand TRAPs from Figure 2c overlaid in red.

Triangles are used frequently to represent the topography of an object, especially in 3D models, and are also used effectively in OBIE. By texture mapping these triangles with corresponding

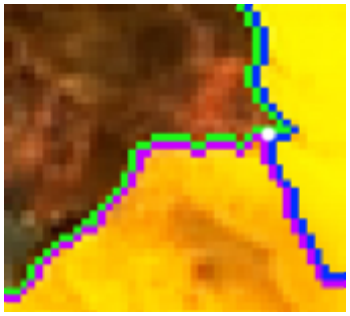


Figure 3: TRAP Boundary Junction. Top left TRAP boundary pixels (green), right TRAP boundary (blue), bottom TRAP boundary (purple). White dot represents three –TRAP junction, where a ridge node is placed for *object* triangulation.

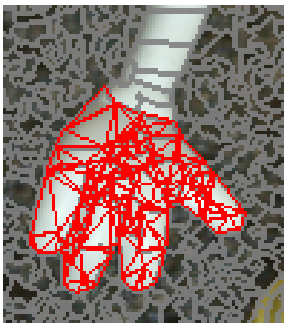


Figure 4: Triangulated TRAPs. Triangles are shown in red, with TRAP boundaries from Figure 2b shown in gray.

regions of the image, we take advantage of OpenGL's hardware acceleration to perform editing operations at interactive rates.

4 OBJECT-BASED EDITING OPERATIONS

After an object has been selected and triangulated, its shape is ready to be edited. Editing operations include traditional linear affine transformations (scale, rotate, translate) as well as *new* non-linear transformations for localized stretching, warping and rotational bend. Objects can be edited directly using gesture motions with the mouse, or indirectly by attaching object geometry to curve deformers. Included also in our suite of editing tools is "object delete" with automatic filling of the space previously occupied by the object.

4.1 Implementation

Efficient implementation of object-based editing operations requires the support of four basic system components: (1) Conversion to, and transformations within a local coordinate system (2) Object and background layering (3) Antialiasing within OpenGL (4) Pivot point placement for localized warping.

For efficiency and simplicity in carrying out otherwise complex object manipulation and editing operations, object vertices are first transformed into a local coordinate system. Every subsequent mouse movement calls the current tool's warping function, which warps the object's temporary vertices in the local coordinate space. Local coordinate transformation coupled with OpenGL rendering of the edited object provides visual update rates of 10-20 frames per second, which is sufficient for interactive object editing.

Second, the selected object is stored on a separate layer from the rest of the image, facilitating overlap with other scene components resulting from the object being pulled and stretched separately from the background.

Third, because the object is stored on its own layer, edges sometimes appear jaggy because of aliasing. Using OpenGL, we provide a simple alpha blending "fix" for the edges of the object. The object polygon is drawn multiple times with decreasing thicknesses into the background (destination) alpha, summing up the alpha into a "ridge" with its peak along the actual boundary. When the object is blended onto the background, edges take on a "feathered" look, providing efficient and reasonable antialiasing (Figure 5). This provides greater degrees of freedom, as compared to [Sander et. al 2000], in the total width of the transition interval (2.5/2 pixels, for example) as well as in the functional form of the transition, as specified by each step increment and its associated alpha value (e.g. .15, .35, ...).

Fourth, the object's pivot point (initially at its center of gravity) is adjustable, and can be moved to any location within the object to allow localized warping by stretching the object with respect to that pivot point.

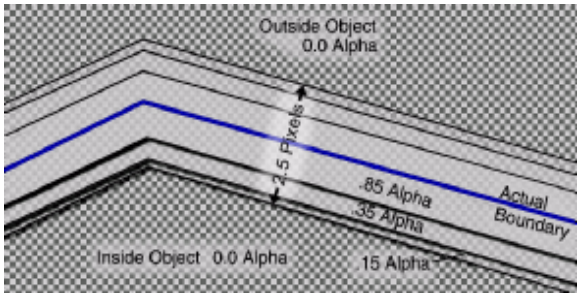


Figure 5: OpenGL Antialiasing of Object Boundary Segment using Alpha Buildup. .(Thick) object outlines straddling the actual object boundary (shown in blue) are drawn onto the background, adding only alpha amounts (and no color values) to each other. Lines of decreasing width are stacked on top of each other, accumulating alpha values. The object is then blended onto the background using the background alpha values to specify how transparent it is when blended. Where the image is most transparent (checkers are most visible) the object will be opaque over the background. At the object boundary, where the alpha value is .85, the object will be almost completely transparent.

4.2 Linear Transformations

Object translation, rotation and scaling are performed with respect to the pivot point in the traditional way, but at interactive rates providing continuous visual feedback. This saves time and eliminates guesswork inherent in other iterative, trial-and-error approaches because the feedback is immediate, and the user can verify the correctness of the desired result *while* the object is being edited.

4.3 Background Filling Behind Objects

When objects are deleted, moved, or otherwise modified in such a way as to leave a hole in the image, the hole needs to be filled somehow. Many methods have been proposed to replicate texture to fill holes, expand borders of an image, and create tileable texture. Efros and Leung presented a fairly successful algorithm [Efros and Leung 1999] that has been built upon in subsequent research [Harrison 2001; Wei and Levoy 2000], accelerating the process from hours down to many seconds. While giving some impressive results for many types of texture, none of these techniques run fast enough to be interactive with normal image sizes.

More recently, many algorithms have surfaced that use regions of texture, rather than computing each pixel separately. There are various methods used for connecting these regions of texture, from alpha channel blending [Xu et al. 2000], to growing the regions together [Efros and Freeman 2001; Praun et al. 2000]. These algorithms are faster, but still require excessive computation and/or a region specifying phase.

Whenever an object's shape is modified, there is a chance that the background behind it will be partly or completely exposed. Since our hole-filling processes run at interactive rates, we simply fill in the entire space behind an object *whenever* it is modified. We have implemented two of many possible algorithms for filling the hole: scale-down filling and random-grid filling. Both methods use texture TRAPs in much the same way as the texture-preserving operations below.

Scale-down filling uses concentric, overlapping displacement of TRAPs to fill toward the center of the object. The scaling is done in steps, the number of which are based on the average background TRAP size. At each step in the scale, copies of all of the background TRAPs are laid down. Figure 6, parts 2a and 3a illustrate this technique.

Random-grid filling creates a grid the size of the object's bounding box, with grid points spaced relative to the average background TRAP size. A random background TRAP is placed at each intersection in the grid, filling the entire grid with overlapping TRAPs. This technique is illustrated in Figure 6, parts 2b and 3b. Since both of these algorithms allow TRAPs to land outside of the object hole, they are again masked off by the shape of the original object.

4.4 Texture-Preserving Operations

When an object is stretched, if its texture is high in detail, the texture becomes overly smoothed, looking unnatural and incorrect. To fix this, we disconnect the object TRAPs, keeping their sizes constant, and warp only their basepoint positions.

Each TRAP is triangulated separately as described in Section 3. Once the TRAP primitives are disconnected, as are the white TRAPs in Figure 7 part 2, rather than warping all of their vertices, we leave them rigid at the same scale, and warp only the positions of their basepoints.

Since spaces will be created between texture TRAPs, we lay down TRAPs continuously with each frame update as we stretch the object, as shown in Figure 7, parts 2 & 3. TRAPs continue to build up in layers as the mouse is moved so that the user can continue to drag the mouse back and forth until the object is sufficiently dense with texture TRAPs. The new TRAPs can be slightly jittered around (moved small random amounts in x and y) to increase randomness in the texture, and can be laid down more sparsely if the density is too thick.

As the region is stretched, the boundary continues to mask off the rest of the stray TRAPs, using a mask shown in Figure 7, part 4. Black areas in the mask represent pixels that will not be drawn to the screen, and correspond to the pixels outside of the object boundary. Jitter amount (extent of the random movements of each TRAP in terms of its bounding box size) and density (percentage of the original TRAPs that will be laid down at each frame), as well as opacity (TRAP transparency), can easily be adjusted with sliders in the user interface.

4.5 Non-linear Transformations

The non-linear stretching and bending tools make use of the local coordinate system transformations discussed earlier and the pivot point placement. The pivot point also acts as a "thumbtack," to stop all movement in the negative x regions of the object's local coordinate system. Figure 8 represents the thumbtack/pivot point as a cyan square near the shoulder of the arm.

When using the stretch tool, the user grabs any point on the object and uses it as a handle to stretch the object with respect to the pivot point. In this way, mouse movement affects both the length and width of the object. Movements in the object's local x direction stretch the object along that x axis for object parts between the handle and the pivot point, and movements in its local y direction increase and decrease the thickness of the object. For any given vertex in the object, its new position (x', y') is computed using

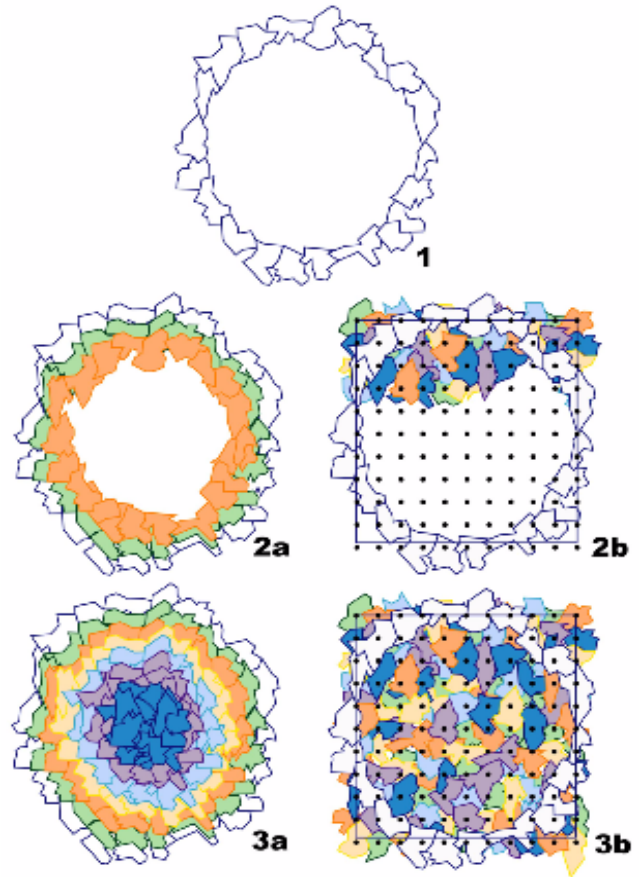


Figure 6: Two Different Hole Filling Techniques. The object hole, showing the immediately surrounding background TRAPs (1) is filled in one of two ways: (2a–3a) Selected background TRAPs are scaled in iteratively, and pasted down at each step. When they reach the center, the hole is filled (3a). (2b–3b) In the second hole-filling approach, selected background TRAPs are randomly placed in a grid-like fashion. The grid spacing is based on the average TRAP size in both X and Y. Pasting TRAPs at each grid location fills the hole (3b). In both approaches, the surrounding background TRAPs mask off the random texture TRAPs to the hole's boundary.

$$x' = x\left(1 + A[i]\frac{\Delta x}{b_x}\right), \qquad y' = y\left(1 + A[i]\frac{\Delta y}{b_y}\right) \tag{1}$$

where $A[i]$ is a general attenuation multiplier, b_x and b_y are the positive x and y dimensions of the object's bounding box, and Δx and Δy are the x and y changes in cursor position. Figure 11 (inset curves) and Figure 13 (bottom) show examples of $A[i]$. An option is also available which allows the object to preserve area as it stretches. As the length increases, the width decreases inversely, and vice versa.

The bend tool essentially rotates the object, but with a (possibly) non-linear attenuation $A[i]$ towards the thumbtack/pivot point, stopping all movement in the negative x regions of the object's local coordinate system. For any given vertex in the object, its new position is computed using

$$(x', y') = \begin{bmatrix} \cos(\theta A[i]) & -\sin(\theta A[i]) \\ \sin(\theta A[i]) & \cos(\theta A[i]) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{2}$$

where $\theta A[i]$ is the *attenuated* rotation angle or the "rotational bend."

An example of rotational bending is shown with Mrs. Potato's arm in Figure 8b. The green outlines show where the arm used to be, and the red line indicates the excursion caused by $\theta A[i]$.

Since it can be more intuitive for a user to stretch an object *as* it is bending, rather than performing the tasks separately, we have combined the two into a single tool. When the cursor-pivot vector is determined for the angle of rotation (Figure 9), the ratio of its length to the length of the original (red) vector is used to compute the stretch factor. Then the stretched position is passed to the rotation computation, which proceeds in the same way as the normal rotation above. For any given vertex in the object, its new position is computed using:

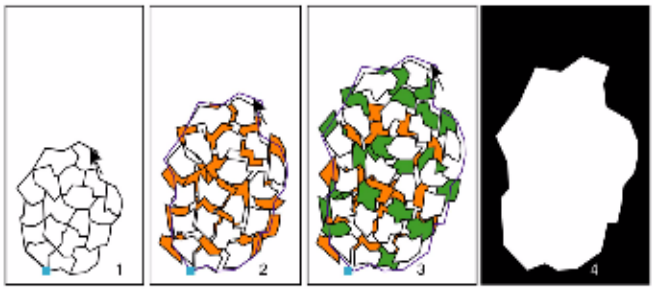


Figure 7: Texture Preservation Diagram. (1) Original object, with TRAPs adjacent to each other, and cursor position as it starts the drag. (2) As the object is expanded, the original (white) TRAPs move with the scale, but stay the same size. They lay down a copy of themselves (orange TRAPs) in a random location near their current position. (3) At each next step of the drag copies (green TRAPs) are laid down near the new positions of each original TRAP. (4) In order to maintain the object boundary while dragging, a binary mask is used in the OpenGL stencil buffer. White/black mask areas allow pixels to be drawn/not drawn, clipping off parts of TRAPs that stray outside the object boundary, such as those in 2 and 3.

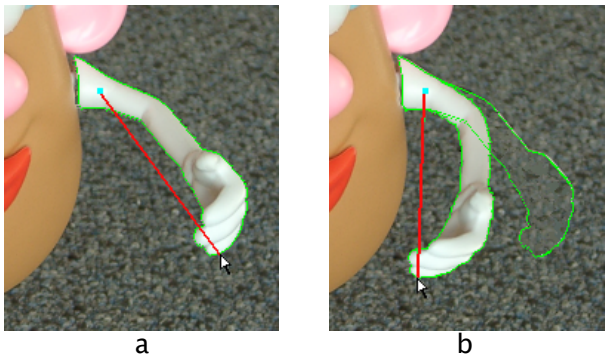


Figure 8: Object Bending. (a) Arm selected (green), anchor point (blue), object axis (red). (b) Rotational bend using cursor movement and $A[i]$.

$$x' = x\left(1 + A_l[i]\frac{v_n - v_o}{b_x}\right), \qquad y' = y \tag{3}$$

$$(x'', y'') = \begin{bmatrix} \cos(\theta A_r[i]) & -\sin(\theta A_r[i]) \\ \sin(\theta A_r[i]) & \cos(\theta A_r[i]) \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{4}$$

where v_n is the cursor-pivot vector length (Figure 9), v_o is the original vector length (red), $A_l[i]$ is the general attenuation multiplier for length, $A_r[i]$ is the attenuation multiplier for rotation and $\theta A_r[i]$ is the *attenuated* rotation angle. Compounding of the attenuation multipliers increases significantly the degrees of freedom with which objects can be stretched and warped.

Figure 9 diagrams the process of stretching and bending an object. This requires two steps computationally, but only a single user interaction. For example, Figure 10 shows the Potato's arm being stretched and bent simultaneously in two different positions, each requiring only a single interaction. The green outlines show where the original arm was in relation to the modified arm. The red lines show the angle of rotational bend.

5 INDIRECT EDITING WITH CURVE DEFORMERS

Section 4 introduced the use of $A[i]$'s, general attenuation multipliers, to modulate the effect of non-linear warping operations. Curve deformers (Figure 11) are one possible way of specifying the $A[i]$'s.

After *direct* editing, additional modifications to the object can be made *indirectly* using curve deformer tools. Curve deformers are implemented using a Bezier curve with four control points to interactively modify the shape of the curve. Control points are constrained to move only in the y direction. The curve is then sampled into a lookup table that defines $A[i]$. After an object has been stretched or bent, selected curves corresponding to length $A_l[i]$, width $A_w[i]$, and rotation $A_r[i]$, can be modified, just as in any drawing application, to reposition, regesture, or otherwise manipulate the object. Figure 11b-f (top) show examples of common curve deformers with their corresponding effect or "fall-off" on the original object in Figure 11a. Notice that while the length of the object may stay the same, the texture of its middle section changes position according to the shape of the corresponding curve deformer.

As control points on the various curves are moved, the object changes shape interactively. Figure 12a shows a close-up similar to Figure 1c, with a default stretch. The hand is somewhat large, which could be useful to show sharp perspective. Figure 12b shows how the thickness curve can be used to shape the already stretched object to a more natural shape and scale.

Any curve deformers, $A_j[i]$ and $A_k[i]$, can be used together. For example, Figure 13a shows the arm bent with the default values for both length and rotation fall-off. The fingers appear distorted and unnaturally large. Figure 13b shows how the length deformer mini-

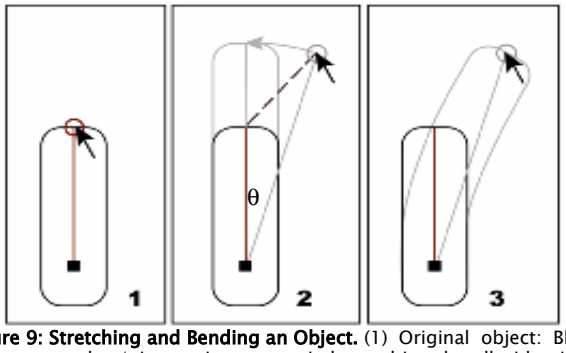


Figure 9: Stretching and Bending an Object. (1) Original object: Black square = anchor/pivot point, open circle = object handle identified with cursor (black arrow), red line = object axis. (2) Dashed line = cursor drag (user interaction), gray lines = computation which amounts to a traditional stretch and rotation of the object through angle θ with the important difference that the stretch and rotation are now attenuated by $A_l[i]$ and $A_r[i]$ (eqs. 3-4). (3) The result is a "rotational bend" obtained from compounding the effects of $A_l[i]$ and $A_r[i]$.

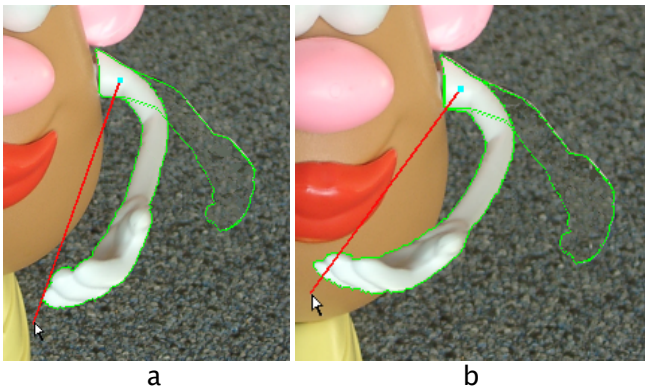


Figure 10: Stretching and Bending Arm. (a) Arm bent and stretched simultaneously using $A_l[i]$ and $A_r[i]$ (see Figure 9) with boundaries (green), axis of rotation (red). (b) Arm bent more to show occlusion.

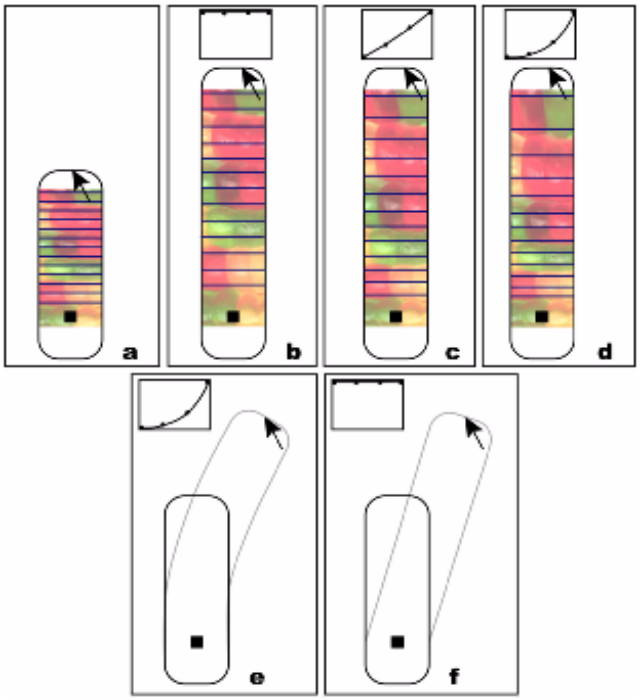


Figure 11: Curve Deformers for Indirect Editing. Examples of how editing the curve shape can modify the object shown in (a). (b-d) show various fall-off amounts for stretching, and (e,f) show two different fall-off amounts for bending.

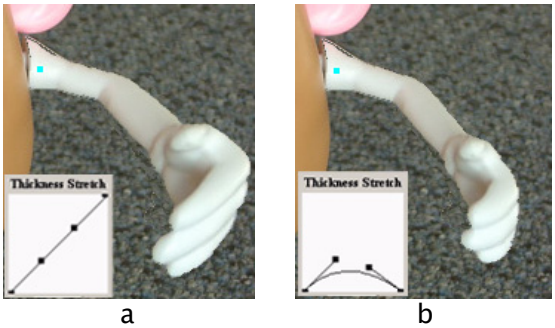


Figure 12: Let's Give Mrs. Potato a Big Hand! (a) The hand stretched with the default thickness values. (b) The same cursor motion, but with different thickness values, especially towards the end of the object (the hand area).

mizes distortion at the end of the object. Because the deformers can work in concert, we can also adjust the rotation fall-off to "tuck" in, and regesture the elbow and forearm area slightly, creating a nicer hand shape as well.

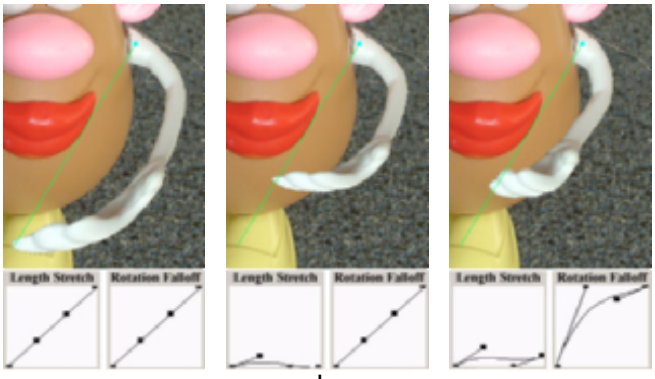


Figure 13: Compounding Curve Deformers. Combining length and rotation fall-off. (a) Default values for length and rotation. (b) Length deformer is adjusted to pull hand back, minimizing distortion. (c) Rotation fall-off deformer is also adjusted to tuck in elbow to improve foreshortening effect, giving user more "gesture" control.

6 TEXTURE PAINTING

Texture painting is performed by "spraying" TRAPs over a region. Various painterly effects are achieved by varying the jitter, opacity, scale and density of the TRAPs (Figure 14).

TRAPs are selected (in groups) and then used as the "paint" for a texture brush. Similar to the process for texture preservation (Figure 7), texture TRAPs are laid down as the brush moves in the image. The texture brush can be used to create new regions from an existing texture. It also constitutes an effective delete tool. In Figure 15 the texture brush accomplishes both purposes.

Figure 15 illustrates the process of painting out a horse from a field. Texture TRAPs are selected in the upper left part of the field (shown with a red outline in Figure 15b), and used to paint out the top part of the horse. More TRAPs are selected in the middle left region (Figure 15c) and used to paint the bottom half. Finally, a region of TRAPs in the lower left corner is selected and used to randomize the seam between the two sections of painted texture, using various amounts of scale and jitter.

Brush options are provided for better artistic usability, and are attached to the GUI by means of sliders. The amount of jitter, opacity and density can be adjusted for the brush. The brush scale can also be adjusted, which changes the size of each TRAP relative to its original selected size. A clone tool provides a similar function to that found in Photoshop, but with the distinctive difference that as the mouse moves to paint, the tethered selection moves, sampling new TRAPs each time with the powerful variations offered in Figure 14. This tool is particularly effective with the jitter and scale options.

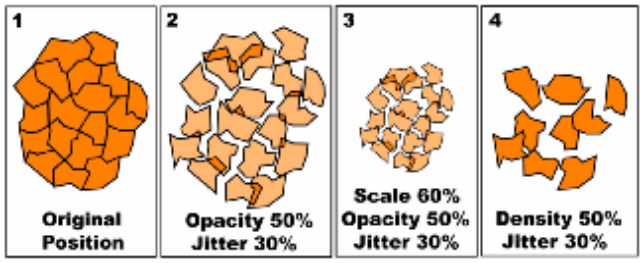


Figure 14: Texture Painting Options. (1) Selected TRAPs in their original positions. (2) TRAPs painted with 50% opacity and 30% jitter. (3) TRAPs painted with 50% opacity, 30% jitter, at 60% of original scale. (4) TRAPs painted with 50% density and 30% jitter.

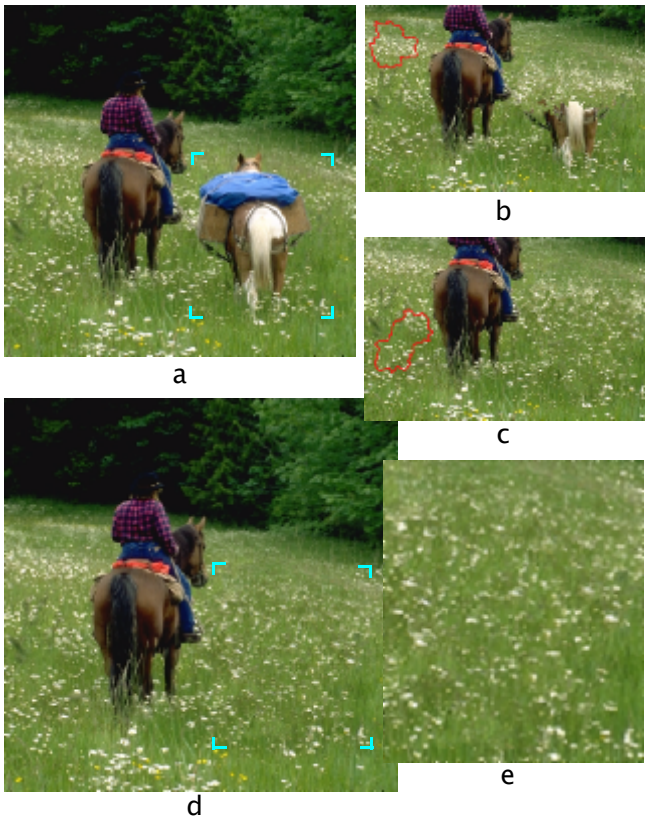


Figure 15: Hard to Find Good Help. (a) Original image. (b) Texture TRAPs sampled from the upper left (red contour) to paint out top half of horse. (c) TRAPs sampled from middle left to paint out bottom half. (d) TRAPs sampled from lower left to give a more random look to the already painted TRAPs. (e) Detail of region of painted texture (blue brackets) where pack horse was originally.

7 RESULTS

Object selection, which typically requires 3-5 seconds, allows OBIE to be applied successfully to a variety of images (Figures 16-25). An Athlon MP 1.2 GHz dual processor, with an nVidia GeForce 3 graphics card was used for all results. Only a few seconds are required for tool and parameter selection. An immense amount of time could be saved in the application of OBIE to 2D animation or claymation (Figures 1,16,20,21). OBIE "extends the reach" (Figure 17) of image editing beyond that available in pixel-based applications.

Figure 22 shows two children "choking a goose," with two other techniques for texturing out the goose shown in (b) and (c). To *really* get rid of the goose using OBIE, we first delete the head and neck, specifying water as the replacement background. Then we select and delete the body and feet, sampling nearby grass as background. This deletion required 10 seconds because it had to be done in 2 steps.

An extended comparison of OBIE with other popular image-editing software, Adobe Photoshop [Adobe 2000], Scansoft's (Kai's) SuperGoo [Scansoft 2001], and Resynthesizer [Harrison 2001], a GIMP plug-in, is given in Figures 19, 22, and 23. The other software often produces reasonable results, but at the expense of noticeable artifacts or excessive user interaction and/or processing time

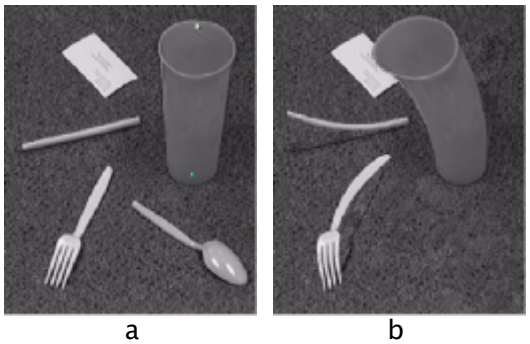
8 CONCLUSIONS

Object-based image editing changes fundamentally the way image editing is performed by changing the granularity of editing operations from the pixel to the object level, and in a way that provides greater control over the object's shape and topography. Individual image objects can be selected and edited, both directly and indirectly, in a

a b c

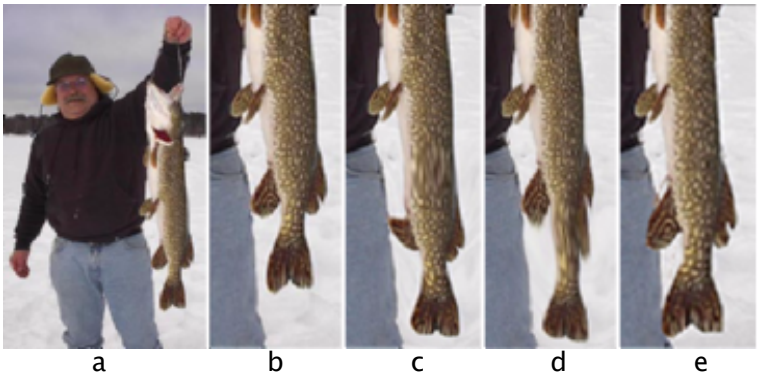
Figure 16: 2D Character Animation. [Smith 2002] (a) Original digitized animation frame. (b) Intermediate frame in sequence. Green outlines show the selected objects' original positions. (c) Arm drooped, flower and hand bent, head and hair drooped, smile bent. Note that the background was filled in all cases as well. Each object required ~5 seconds to select and 5-10 seconds to bend/move, while preserving frame-to-frame color coherence. Total time per keyframe was ~2 minutes compared with ~30 minutes to draw by hand.

Figure 17: Beach Scene. Both the towel and the boy's arm are warped with the bend-stretch tool [insets]. "Background sand" automatically fills object "holes."



a b

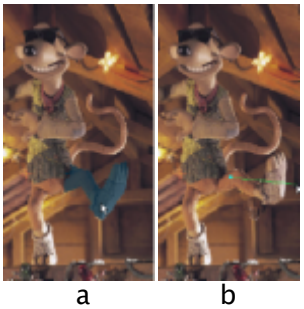
Figure 18: Desktop Editing. (a) Original scene with various common items. (b) Cup, pen and fork are bent, and spoon is painted out with the texture brush.



a b c d e

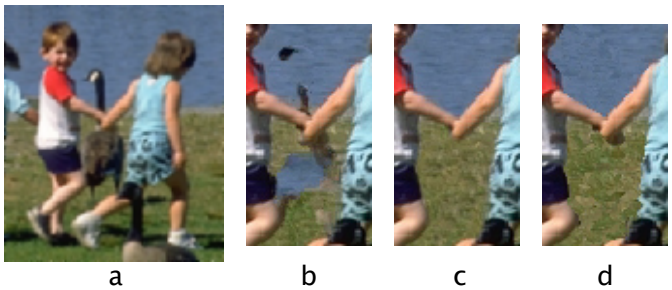
Figure 19: "And it was THIS Big!" (a) Original image. (b) Scales scaled. (c) Photoshop 19 seconds - smeared snow, jeans and scale texture in parts. (d) SuperGoo - 18 seconds - similar smearing problems. (e) OBIE - 2 seconds - stretched the fish and filled holes automatically with 1 user-specified sample region of snow TRAPs, leaving snow and jeans as they were.

Figure 20: "Smile!" Cursor: Straight-faced chicken. Inset: Corners of mouth are pulled up separately using OBIE.



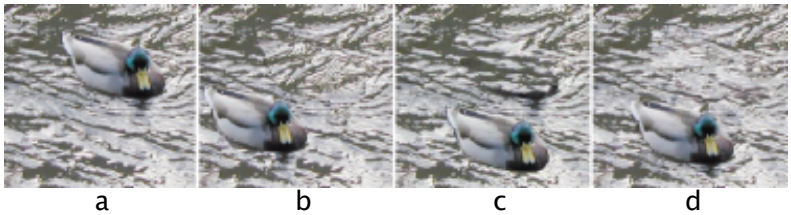
a b

Figure 21: Dancin' in the Rafters. (a) Mouse foot selected (blue). (b) Foot is bent and stretched to a different angle using OBIE. Space originally occupied by foot is filled automatically.



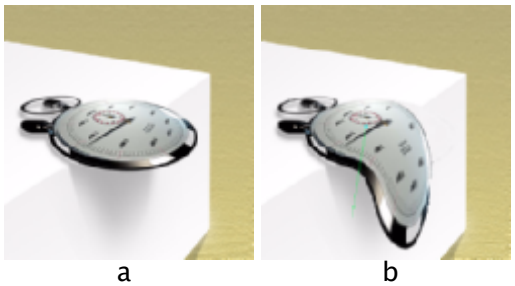
a b c d

Figure 22: (a) Goose choking fun (b) Resynthesizer fix - 8 min. (c) Photoshop - 3 min., 10 samples (d) OBIE - 10 sec., 2 samples.



a b c d

Figure 23: The Water Hole. (a) Original image. Goal is to fill in the space (hole) where the duck was. (b) Photoshop - 28 seconds - required only one sample (clone) point and water appears copied into the hole. (c) Resynthesizer - 17 minutes - less convincing. (d) Duck moved with OBIE, and hole filled automatically in 6 seconds, with 1 user-specified sample region of water.



a b

Figure 24: To Dali-ize. (a) Original image. (b) SIGGRAPH droop ("time warp").

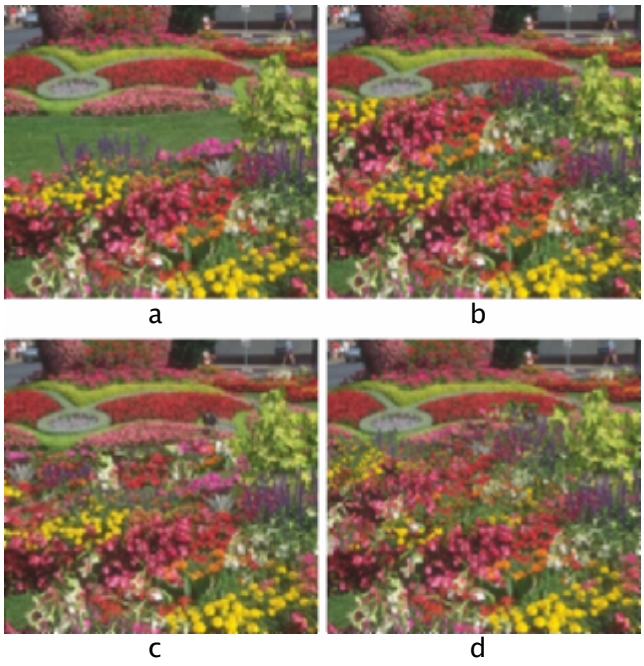


Figure 25: Extending Flower Beds. (a) Original image. Goal is to extend foreground garden across lawn to meet hill, using clone tools. (b) Photoshop – 10 seconds, 1 sample (clone region); flowers look like a direct copy and are same scale as closer ones. (c) Resynthesizer requires map creation for texture sampling. Total time to fill texture (including map making) – 20 minutes. (d) OBIE – 10 seconds, 1 starting user-specified sample region of flower TRAPs. Note slightly smaller back layer appears more distant, demonstrating more natural variation.

few seconds to a few 10's of seconds, in ways not possible or practical using pixel-based editing or polygonal warping.

The visual quality of edited images compares favorably with images edited using Photoshop, GIMP's Resynthesizer plug-in, or SuperGoo. Additionally, tedious tasks such as selecting new sample points for Photoshop's clone tool, and painting to fill in holes, are largely or completely automated. Time taken to perform many simple tasks is shortened dramatically, requiring a matter of seconds, compared with a matter of minutes or hours using Photoshop or Resynthesizer. Object manipulation and texture painting/hole-filling (for stochastic textures) are now interactive tasks, giving the user important visual feedback *during* mouse interaction.

Object selection fails (rarely) when image gradients completely disappear at perceived object boundaries, because object and background pixels combine into a single TRAP. While this can be overridden manually, other techniques for object segmentation and decomposition may overcome this automatically and provide a coarser, more appropriate level of subobject detail.

One area that needs significant improvement is background filling. Concentric and gridded filling work reasonably well for stochastic textures but struggle with complex backgrounds composed of regular or well-defined structures. Filling to a medial axis and exploiting recent work in texture synthesis could significantly improve background filling. Intelligent Paint or Intelligent Scissors with sub-pixel accuracy [Mortensen and Barrett 1999] could also be used to remove object fringe from background (see Figure 10).

While a single anchor point provides significant flexibility in object deformation, we would like to also introduce anchor lines or curves, and allow the use of multiple anchor points simultaneously.

We would also like to extend curve deformers to include other object shape properties (area, eccentricity, etc.) - all compoundable, and allow curve deformers to affect object shape independent of any initial stretch or scale. Depth-ordered object layers could also be used.

If the user desires to warp an image on a global scale with smooth continuity, OBIE tools would not be the correct choice, because pieces will start to break apart. However, for object-level control, OBIE has much to offer over current pixel-based methods.

Grateful Acknowledgement to Dreamworks SKG and Aardman Animations for permission to use frames from Chicken Run©2000.

References

ADOBE SYSTEMS INCORPORATED 2000. Adobe Photoshop Version 6.0 User Guide.

BEIER, T. AND NEELY, S. 1992. Feature Based Image Metamorphosis. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92), 26(2)*, ACM, 35-42.

BERN, M. AND EPPSTEIN, D. 1992. Polynomial-size nonobtuse triangulation of polygons. *International Journal of Computational Geometry and Applications 2(3)*, 241-255.

BOOKSTEIN, F. L. 1989. Principal Warps: thin-plate splines and the decomposition of deformations. In *IEEE Transactions on PAMI, 11(6)*, 567-585.

EFROS, A. A. AND LEUNG, T. K. 1999. Texture Synthesis by Non-parametric Sampling. In *IEEE International Conference on Computer Vision (ICCV 99)*, Corfu, Greece.

EFROS, A. A. AND FREEMAN, W. 2001. Image Quilting for Texture Synthesis and Transfer. In *Proc. of ACM SIGGRAPH 2001*, ACM Press/ACM SIGGRAPH, New York. E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 341-346.

ELDER, J. H. AND GOLDBERG, R. M. 1998. Image Editing in the Contour Domain. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 98)*, 374-281.

GAO, P. AND SEDERBERG, T. W. 1998. A work minimization approach to image morphing, *The Visual Computer 14*, 390-400.

THE GIMP. http://www.gimp.org.

HARRISON, P. 2001. A Non-hierarchical Procedure for Re-synthesis of Complex Textures. In *Proceedings of Winter School of Computer Graphics 2001*, 190-197.

HASBRO, INC. 2001. Mrs. Potato Head. http://www.hasbropreschool.com/

MCINERNEY, T. AND TERZOPOULOS, D. 2000. T-Snakes: Topology Adaptive Snakes. In *Medical Image Analysis, Vol. 4,* 73-91.

MORTENSEN, E. N., AND BARRETT, W. A. 2001. A Confidence Measure for Boundary Detection and Object Selection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2001), Vol. 1*, 477-484.

MORTENSEN, E. N. AND BARRETT, W. A. 1995. Intelligent Scissors for Image Composition. In *Proceedings of ACM SIGGRAPH 1995*, ACM Press/ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 191-198.

MORTENSEN, E. N. AND BARRETT, W. A. 1998. Interactive Segmentation with Intelligent Scissors. In *Graphical Models and Image Processing, 60(5)*, 349-384.

MORTENSEN, E. N. AND BARRETT, W. A. 1999. Toboggan-Based Intelligent Scissors with a Four Parameter Edge Model. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 99)*, 452-458.

MORTENSEN, E. N., REESE, L. J., AND BARRETT, W. A. 2000. Intelligent Selection Tools. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition '99. Vol. II*, 776-777.

MOUNT, D. M., AND SAALFELD, A. 1988. Globally-equiangular Triangulations of Co-circular Point in O(n log n) Time. In *Proceedings of the 4th Symposium on Comp. Geometry*, ACM.

PRAUN, E., ET AL. 2000. Lapped Textures. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press/ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 465-470.

REESE, L. J. 1999. *Intelligent Paint: Region-Based Interactive Image Segmentation.* Masters Thesis, Department of Computer Science, Brigham Young University, Provo, UT.

SANDER, P. V., ET AL. 2000. Silhouette Clipping. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press/ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 327-334.

SCANSOFT, INC. 2001. Kai's SuperGoo. http://www.scansoft.com/products/goo.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. *First Workshop on Applied Computational Geometry*, ACM, 124-133.

SIBSON, R. 1978. Locally equiangular triangulations. *Computer Journal, 21*:243-245.

SMITH, M. D. 2002. Sally's Flower. Graphics Lab, Brigham Young University. mike_d_smith@byu.edu. Received by personal communication.

VINCENT, L. AND SOILLE, P. 1991. Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations. In *IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(6)*:583-598.

WEI, L. AND LEVOY, M. 2000. Fast Texture Synthesis using Tree-structured Vector Quantization. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press/ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, ACM, 479-488.

WOLBERG, G. 1998. Image Morphing: a Survey, *The Visual Computer 14*, 360-372.

XU, Y., ET AL. 2000. Chaos Mosaic: Fast and Memory Efficient Texture Synthesis. *Technical Report MSR-TR-2000-32*, Microsoft Research.

YU, XIAOHUA, MORSE, B. S., AND SEDERBERG, T. W. 2001. Image Reconstruction Using Data-Dependent Triangulation. *IEEE Computer Graphics and Applications Vol. 21, No. 3*, 62-68.