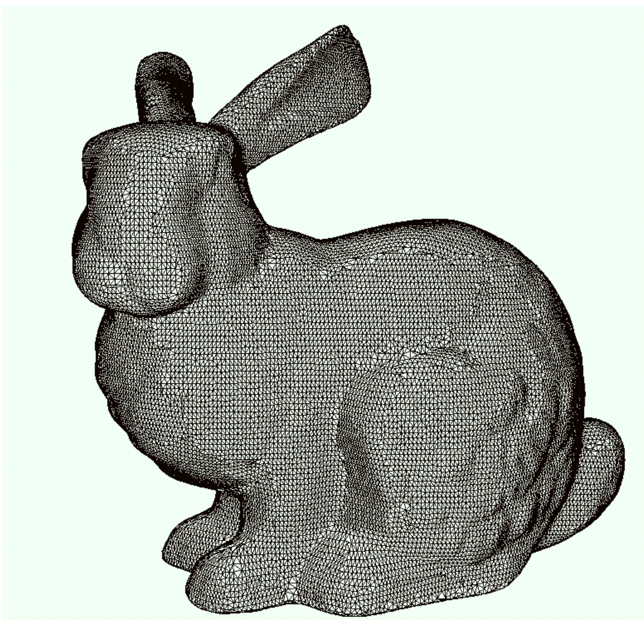
Surface Simplification Algorithms Overview

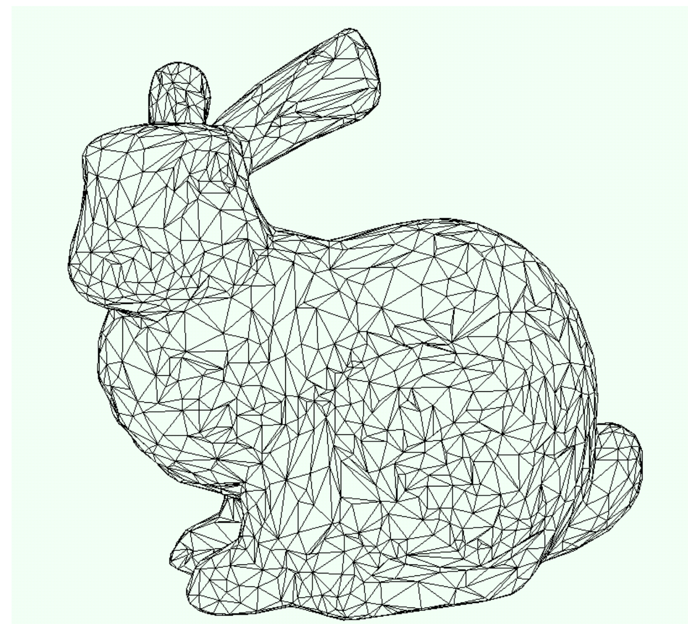
Surface Simplification

- Problem statement:

  given a surface [described with a triangular mesh], find an approximation mesh which minimizes both the **size** and the **approximation error**
- Main issues: **speed, precision, robustness and generality**



70K triangles



2K triangles

Simplification Algorithms

Simplification approaches:

- incremental methods based on local updates
  - **mesh decimation** [Schroeder et al. `92, ... + others]
  - **energy function optimization** [Hoppe et al. `93, Hoppe ‘96, Hoppe ‘97]
  - **quadric error metrics** [Garland et al. ’97]
- coplanar facets merging [Hinker et al. `93, Kalvin et al. `96]
- re-tiling [Turk `92]
- clustering [Rossignac et al. `93, ... + others]
- wavelet-based [Eck et al. `95, + others]

Incremental methods based on ***local updates***

- All of the methods such that :
  - simplification proceeds as a sequence of ***local updates***
  - each update ***reduces mesh size*** and [monotonically] ***decreases*** the ***approximation precision***

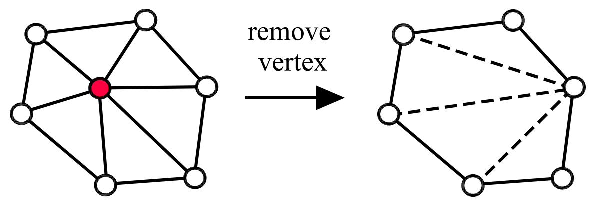
- Different approaches:
  - **mesh decimation**
  - **energy function optimization**
  - **quadric error metrics**

... Incremental methods based on ***local updates ...***

- Local update actions:

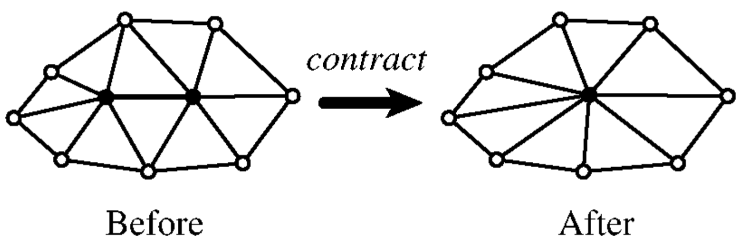
No. Faces

- **vertex removal**



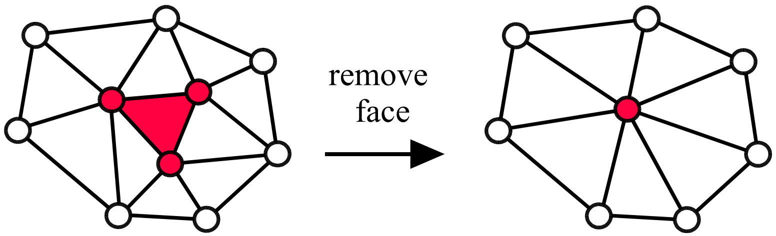
n-2

- **edge collapse**
  - **preserve location**
  - **new location**



n-2

- **triangle collapse**
  - **preserve location**
  - **new location**



n-4

... Incremental methods based on ***local updates ...***

The common framework:

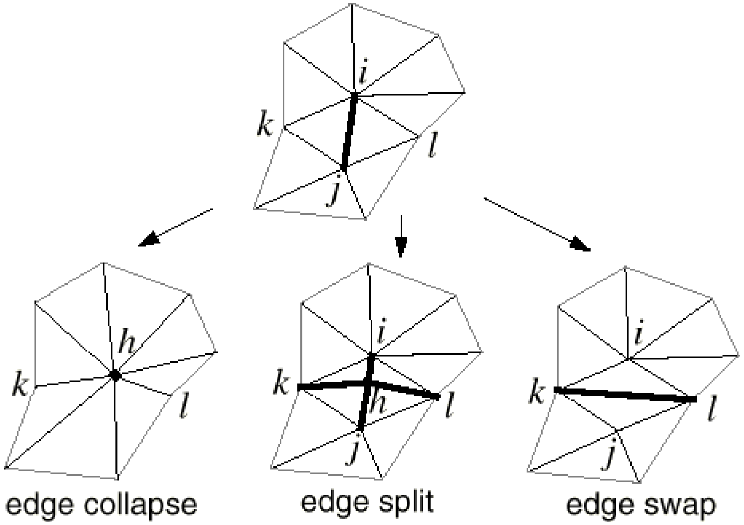
- **loop**
  - ***select*** the element to be deleted/collapsed;
  - ***evaluate approximation*** introduced;
  - ***update*** the mesh after deletion/collapse;

until mesh **size/precision** is satisfactory;

Energy function optimization

Mesh Optimization **[Hoppe et al. `93]**

- Simplification based on the iterative execution of :
  - edge collapsing
  - edge split
  - edge swap



... Energy function optimization: Mesh Optimization ...

- approximation quality evalued with an **energy function** :

$$E(M) = E_{dist}(M) + E_{rep}(M) + E_{spring}(M)$$

which evaluates geometric **fitness** and repr. **compactness**

$\mathbf{E_{dist}}$: sum of squared distances of the original points from M

$\mathbf{E_{rep}}$: factor proportional to the no. of vertex in M

$\mathbf{E_{spring}}$: sum of the edge lenghts

Algorithm structure

- outer minimization cicle (***discrete*** optimiz. probl.)
  - choose a legal action (edge collapse, swap, split) which reduces the energy function
  - perform the action and update the mesh (M_i -> M_{i+1})

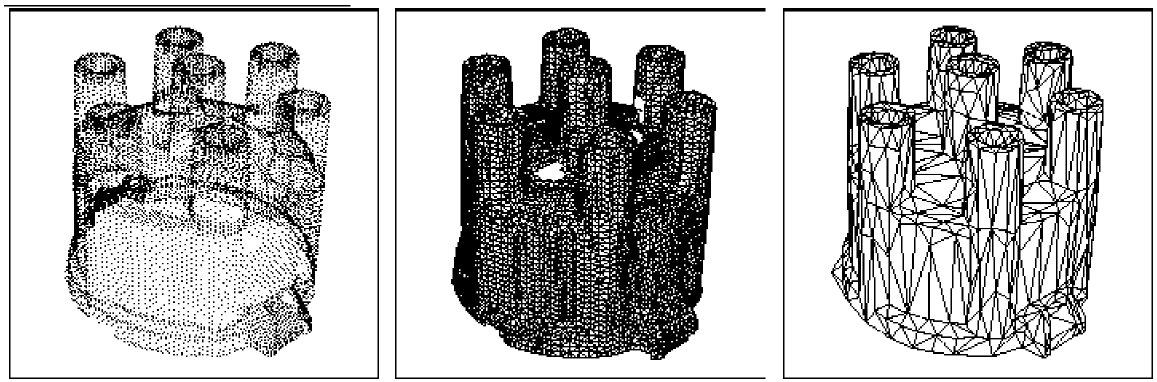
- inner minimization cicle (***continuous*** optimiz. probl.)
  - optimize the vertex positions of M_{i+1} with respect to the initial mesh M_0

but (to reduce complexity)

- legal action selection is random
- inner minimization is solved in a fixed number of iterations

... Energy function optimization: Mesh Optimization ...

Mesh Optimization - *Examples*



(j) Laser range data ($n = 12,745$) (k) Output of phase one (l) Output of phase two

[Image by Hoppe et al.]

Mesh Optimization - *Evaluation*

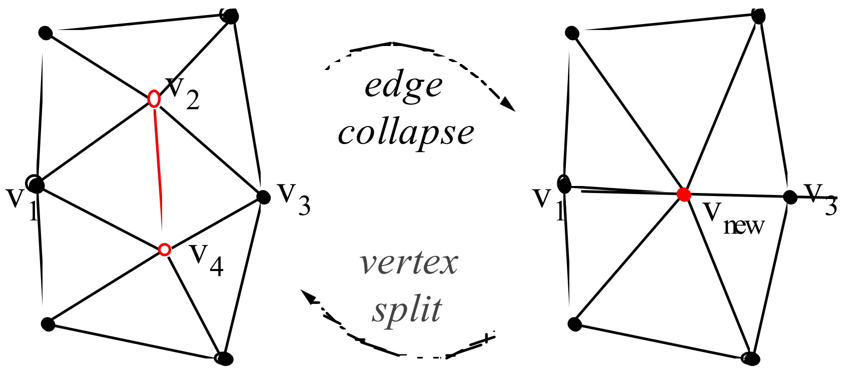
- high quality of the results
- preserves topology, re-sample vertices
- high processing times
- not easy to implement
- not easy to use (selection of tuning parameters)
- adopts a global error evaluation, but the resulting approximation is not bounded

implementation available on the web

... Energy function optimization: **Progressive Meshes** ...

Progressive Meshes **[Hoppe `96]**

- execute **edge collapsing *only*** to reduce the *energy function*
- *edge collapsing* can be easily inverted ==> store sequence of inverse *vertex split* trasformations to support:
  - multiresolution
  - progressive transmission
  - selective refinements
  - geomorphs

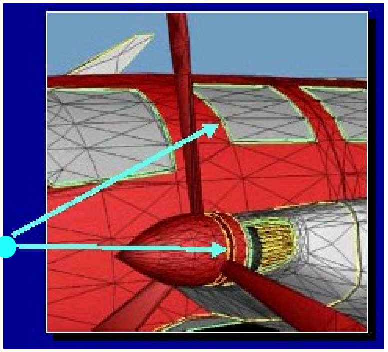


- *faster* than MeshOptim.

... Energy function optimization: **Progressive Meshes** ...

Preserving mesh ***appearance***

- shape and crease edges
- scalar fields discontinuities (e.g. color, normals)
- discontinuity curves



[image by H. Hoppe]

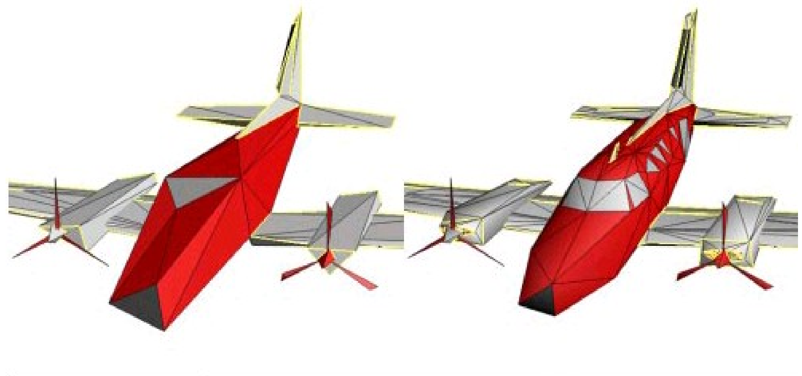
Managed by inserting two new components in the *energy function*:

- E_{scalar}: measures the accuracy of scalar attributes
- E_{disc}: measure the geometric accuracy of discontinuity curves

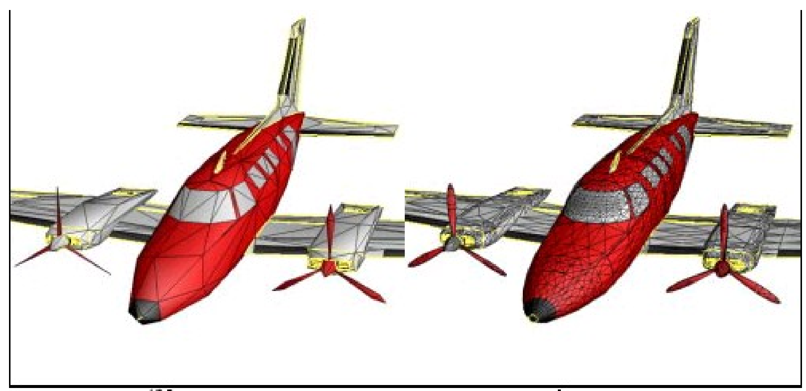
... Energy function optimization: **Progressive Meshes** ...

Progressive Meshes

Examples



(a) Base mesh M^0 (150 faces) (b) Mesh M^{175} (500 faces)



(c) Mesh M^{425} (1,000 faces) (d) Original $\hat{M} = M^n$ (13,546 faces)

... Energy function optimization: **Progressive Meshes**...

Progressive Meshes - *Evaluation*

- high quality of the results
- preserves topology, re-sample vertices
- not easy to implement
- not easy to use (selection of tuning parameters)
- adopts a global error evaluation, not-bounded approximation
- preserves vect/scalar attributes (e.g. color) **discontinuities**
- supports **multiresolution** output, geometric morphing, **progressive transmission, selective** refinements
- much **faster** than MeshOpt.

will be available in MS DirectX 5.0 graphics interface

Decimation

Mesh Decimation

[Schroeder et al'92]

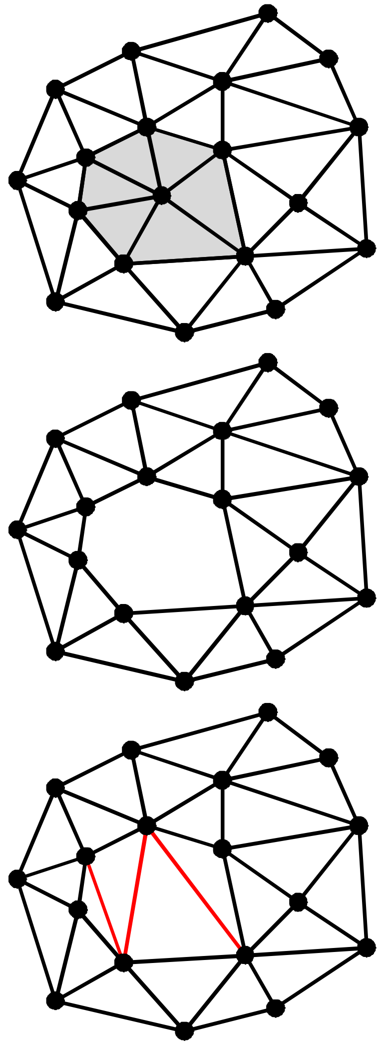
- Based on controlled removal of ***vertices***
- Classify vertices as ***removable*** or ***not*** (based on local topology / geometry and required precision)

Loop

- choose a *removable* vertex v_i
- delete v_i and the incident faces
- re-triangulate the hole

until

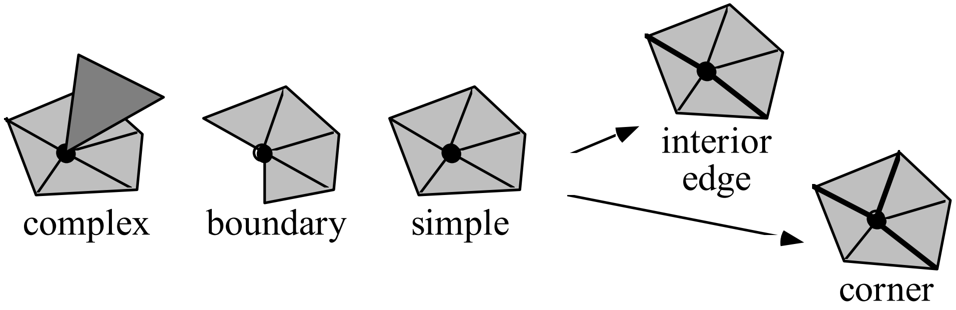
no more removable vertex **or** reduction rate fulfilled



- General method (manifold/non-manifold *input*)
- Algorithm phases:
  - topologic classification of vertices
  - evaluation of the decimation criterion (error evaluation)
  - re-triangulation of the removed triangles patch

Topologic classification of vertices

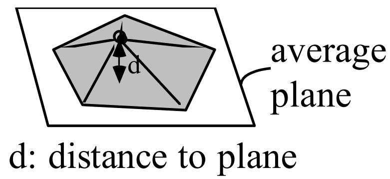
- for each vertex: find and characterize the loop of incident faces



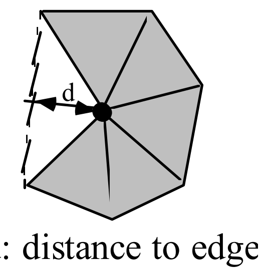
- *interior edge*: if dihedral angle between faces $< k_{angle}$
  (k_{angle} : user driven parameter)
- *not-removable vertices*: complex, [corner]

Decimation criterion -- a vertex is *removable* if:

- **simple** vertex:
  if distance ***vertex - face loop average plane***
  is lower than ε_{max}



- **boundary / interior / corner** vertices:
  if distance ***vertex - new boundary/interior edge***
  is lower than ε_{max}

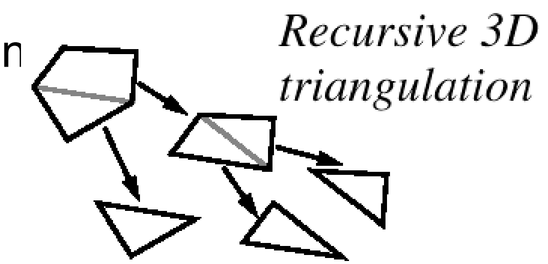


- adopts *local evaluation* of the approximation!!
- ε_{max} : value selected by the user

S I M P L I F I C A T I O N | A L G O R I T H M S

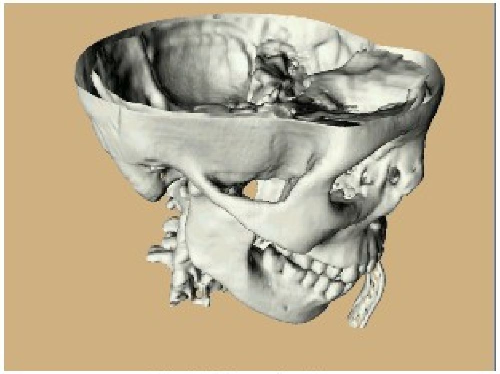
Re-triangulation

- face loops in general non planar ! (but star-shaped)
- adopts ***recursive loop splitting*** re-trian

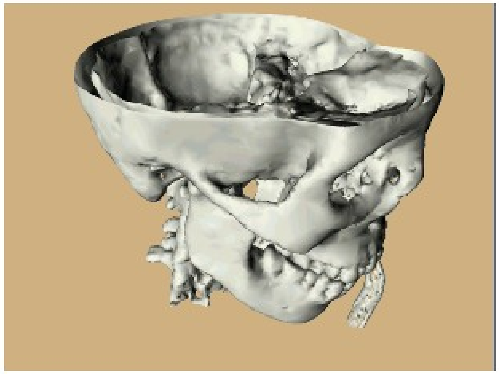


- control *aspect ratio* to ensure simplified mesh quality
- for each vertex removed:
  - *if* simple or boundary vertex ==> 1 loop
  - *if* interior edge vertex ==> 2 loops
  - *if* boundary vertex ==> - 1 face
  - *otherwise* ==> - 2 faces

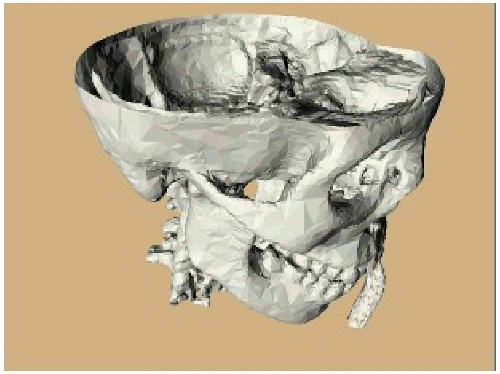
Decimation - *Examples*



Full Resolution
(569K Gouraud shaded triangles)



75% decimated
(142K Gouraud shaded triangles)



75% decimated
(142K flat shaded triangles)

90% decimated
(57K flat shaded triangles)

(images by W. Lorensen)

S I M P L I F I C A T I O N | A L G O R I T H M S

Original Mesh Decimation - *Evaluation*

- good efficiency (speed & reduction rate)
- simple implementation and use
- good approximation
- works on huge meshes
- preserves topology; vertices are a subset of the original ones
- error is ***not*** bounded (local evaluation ==> accumulation of error!!)

*implemented in the **Visualization Toolkit** (**VTK**), public domain*

Enhancing Mesh Decimation

- Improve approximation precision, ensure bounded error
  - **bounded** error [Cohen'96, Gueziec'96]
  - **global error** evaluation [Soucy'96, Bajaj'96,Klein'96,Ciampalini'97, +...]
  - smarter **re-triangulation** (edge flipping) [Bajaj'96, Ciampalini'97]
- Multiresolution, dynamic LOD [Ciampalini'97]
- Decimate other entities
  - **edges** (collapse into vertices) [Gueziec'95-'96,Ronfard'96, Algorri96]
  - **faces** (collapse into vertices) [Hamann '94]
- Preserve color and attributes info
  [Soucy'96, Cohen et al 98, Cignoni etal 98, +….]
- Topology simplification [Lorensen 97]
- Extension to 3D meshes (tetrahedral meshes)
  [Renze'96, Trotts etal 98, Staadt et al 98]

Approximation Error Evaluation

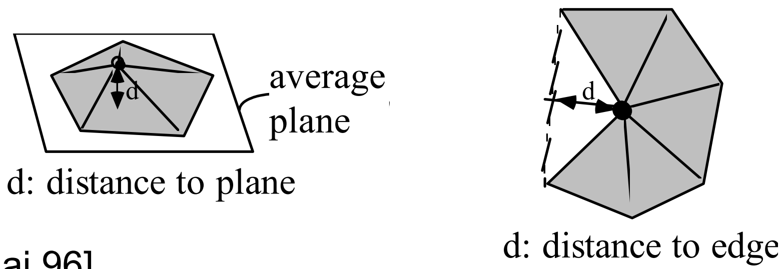
Classification of simplification methods based on ***approximation error*** evaluation euristics:

User' viewpoint:
- simple to grasp
- simple to drive

- **locally-bounded** error, based on mesh distances [ex. standard Mesh Decimation] → *very handy*
- **globally bounded** error, based on mesh distances [ex. Envelopes + enhanced Decimation + others] → *very handy*
- control based on **mesh characteristics** [ex. vertex proximity, mesh curvature] → *may be misleading*
- **energy function** evaluation [ex. Mesh Optim. , Progr. Meshes] → *not easy, many parameters to be selected*

Heuristics proposed for *local error evaluation*:

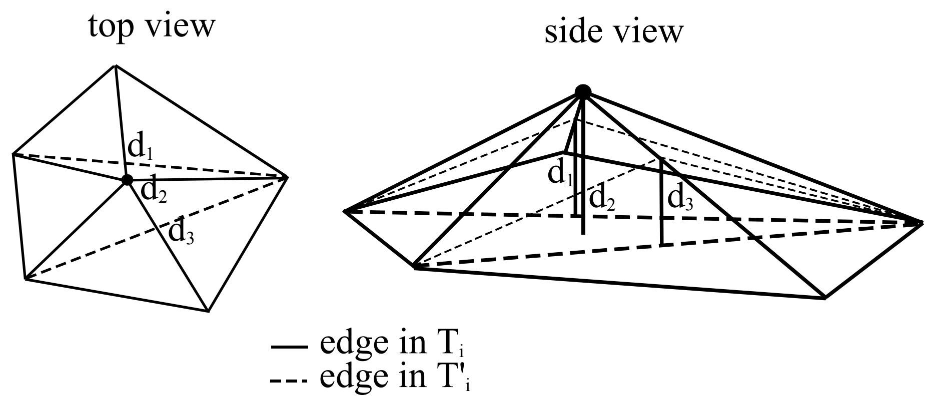
- *approximate* evaluation [Schroeder 92]



- *correct* evaluation [Bajaj 96]

  given two linear patches ==>

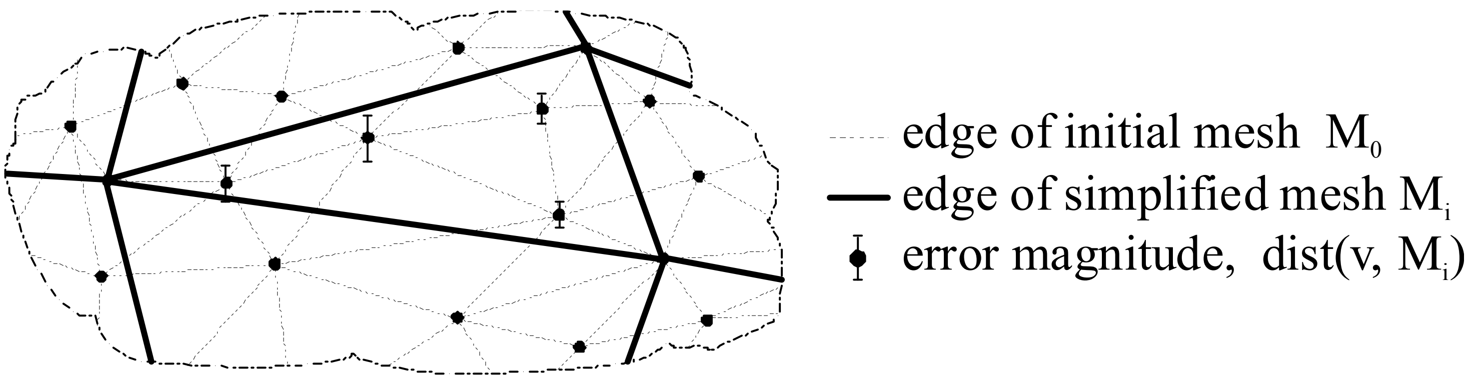
  the max value of meshes' distance is either on edges' intersections or on internal vertices



Heuristics proposed for ***global error evaluation***:

- ***accumulation*** *of local errors* [Ciampalini97]
  fast, **but** approximate

- ***vertex--to--simplified mesh* distance** [Soucy96]
  requires storing which of the original vertices maps to each simplified face;
  very near to exact value (but large under-estimation in the first steps)



SIMPLIFICATION | ALGORITHMS

… Heuristics proposed for ***global error evaluation***:

- ***input mesh -- to -- simplified mesh <u>edges</u>*** **distance** [Ciampalini97]
  - for each internal edge:
    - select sampling points $\mathbf{p}_i$ (regularly/random)
    - evaluate distance $d(M_0, \mathbf{p}_i)$

  sufficiently precise and efficient in time

- ***input mesh -- to -- simplified mesh*** **distance** [Klein96]

  precise, **but** more complex in time

- **use *envelopes*** [Cohen et al.'96]

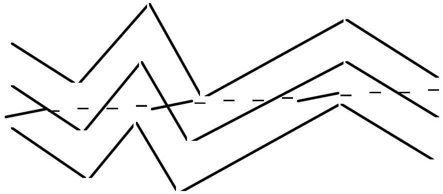
  precise, no self-intersections **but** complex in time and to be implemented

Simplification Envelopes

[Cohen et al.'96]

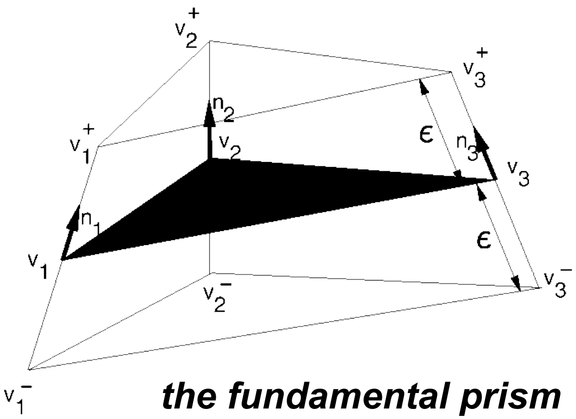
- given the input mesh ***M***
  - build two envelope meshes M_- and M_+ at distance **-ε** and **+ ε** from ***M*** ;
  - simplify M (following a decimation approach) by enforcing the decimation criterion:

    a candidate vertex may be removed **only if** the new triangle patch does not intersect neither M_- or M_+



- by construction, envelopes do not self-intersect

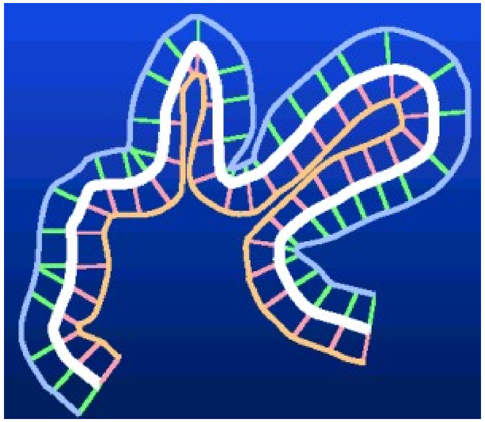
  ==> simplified mesh is **not self-intersecting** !!



the fundamental prism

- distance between envelopes becomes smaller near the bending sections, and simplification harder

- **border tubes** are used to manage open boundaries



(drawing by A. Varshney)

Simplification Envelopes - *Evaluation*

- works on manifold surface **only**
- bounded approximation
- construction of envelopes and intersection tests are not cheap
- **>** three times more RAM (input mesh + envelopes + border tubes)
- preserve topology, vertices are a subset of the original, prevents self-intersection

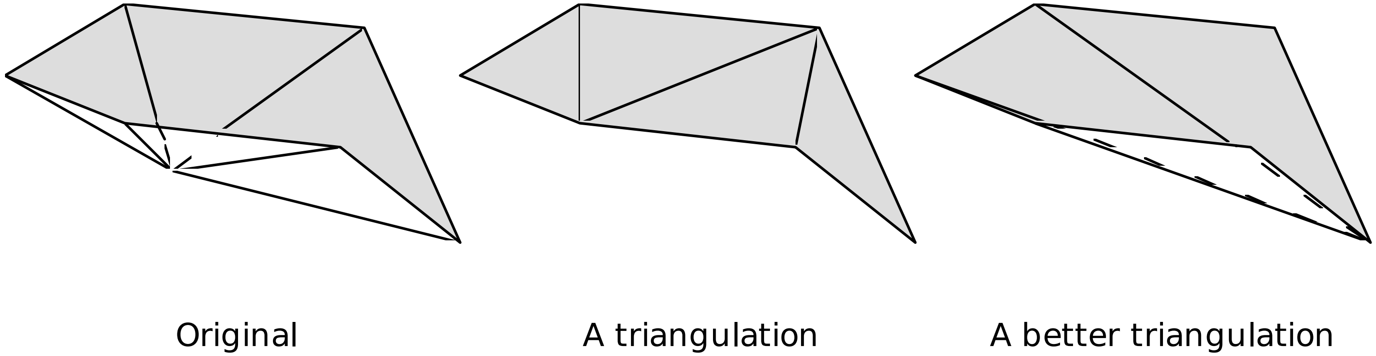
available in public domain

SIMPLIFICATION | ALGORITHMS

For all methods based on re-triangulation, approximation depends on **new patch quality**

- control new triangles' **aspect ratio**, to avoid slivery faces [equiangularity]

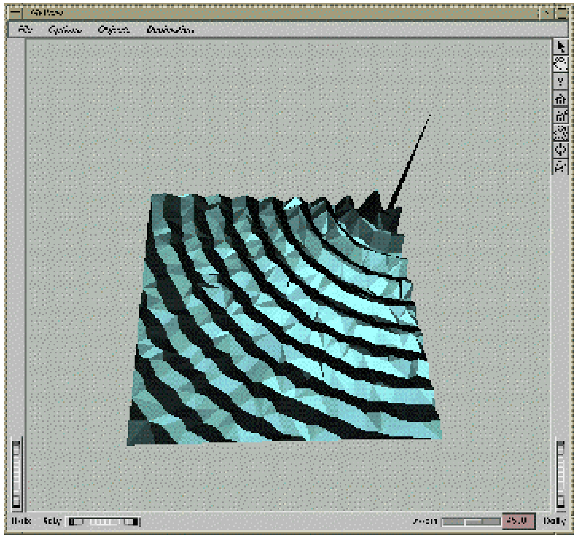
- adopt **edge flipping** to improve mesh quality [Bajaj'96, Ciampalini97]
  - build a first triangulation and, through a **greedy** optimization process based on edge flipping, adapt it to the original mesh
  - **global error** estimate is needed to support flipping



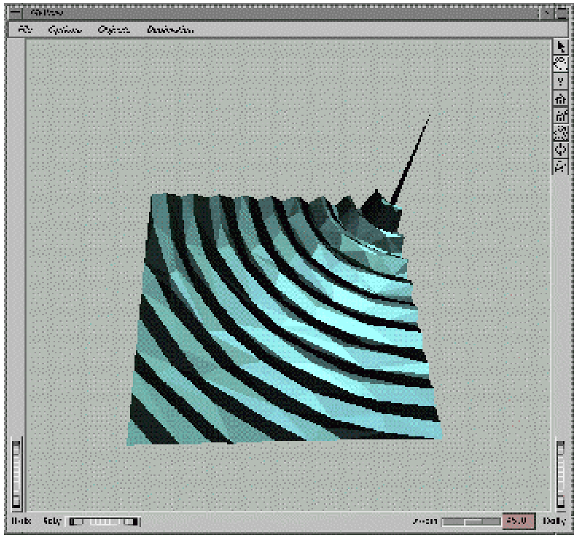
Mesh approximation improvement due to edge flipping **(Jade2.0 code)**

- original mesh: 28,322 triangles
- simplified meshes: same approximation error

no flipping: 1004 faces



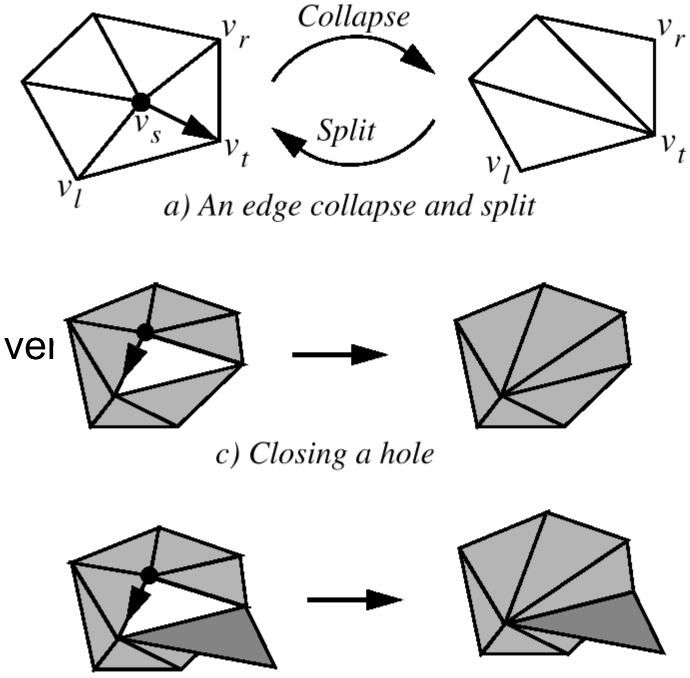
with flipping: 528 faces



Topology Modifying Progressive Decimation

[Schroeder Vis97]

- *topology preservation*: a limiting factor in overall reduction capability
- adopts a **progressive-mesh** approach on top of an **edge-collapse** based mesh decimator
- atomic action: **edge collapse** encoded for progressive storage, transmission, and reconstruction **==>** holes may close, non-manifold attachments may form
- uses a priority queue to store candidate ver
- *available in the **vtk** system*



a) An edge collapse and split

c) Closing a hole

d) Forming a non-manifold attachment

Jade 2.0 (Multiresolution Glob. Err. Decim

[Ciampalini et al.'97]

- **Goals:**
  - ***Speed***
  - ***Precision*** global error management
  - ***Simpl. Efficiency*** good compression ratio
  - ***Generality*** not orientable, not manifold surfaces
  - ***Multiresolution output***
  - ***Ease of use*** given a target **# vertices** ==> "best" quality mesh given a target **approx.error** ==> "smallest" mesh
- **code in the public domain** (SGI executables only)

Candidate vertices selection

- *vertex* ***classification***: same as standard Mesh Decimation
- uses an **heap** to store candidate vertices in order of error
  - *heap initialization*: for all vertices, simulate removal and evaluate approximation introduced
- evaluation of the ***error*** introduced while removing a vertex:
  - approximated *input_mesh---to---simpl_mesh* distance
  - integrated with *edge flipping* test
- *vertex* ***selection*** for removal:
  - in order of **increasing error** (from ***heap***)
  - decimating sorted vertices improves mesh quality and is crucial to support multiresolution

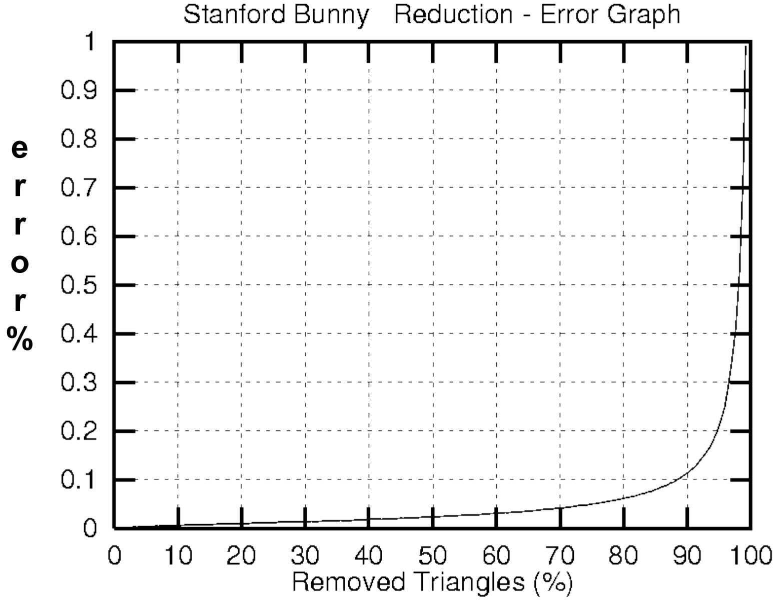
SIMPLIFICATION | ALGORITHMS

```
Algorithm Jade 2.0 (M0,target_err,S)
    Var VH: Heap;    \{vertex heap, sorted by increasing error\}
      S:= M0;

      \{initialize the heap VH: \}
      FOR EACH vertex vi in M0 DO
            compute error ei associated to the removal of vi (includes
    re-triangulation but not mesh update);
    insert (vi,ei) in VH;

      \{main cycle: \}
      REPEAT
    pop first candidate v from VH;
    delete from simplified mesh S;
    retriangulate the hole in S;
    err := current_approximation(S);
    check error in VH for the vertices on the border of the
    re-triangulated hole (and, in case, update heap VH );
      UNTIL err <= target_err;
    END;
```

S I M P L I F I C A T I O N | A L G O R I T H M S

Results

- Simplification times ~= linear with mesh size



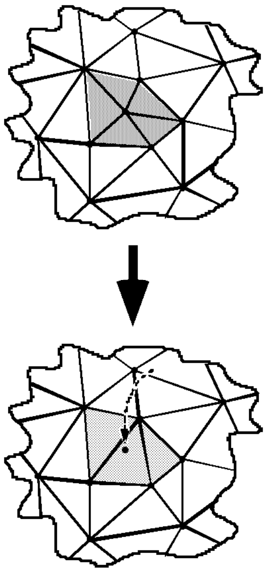
no staircase abrupt error increase

(fundamental for the quality of the multiresolution output)

Construction of a multiresolution model

Keep the ***history*** of the simplification process :

- when we remove a vertex we have **dead** and **newborn** triangles
- assign to each triangle *t* a ***birth error*** t_b and a ***death error*** t_d equal to the error of the simplified mesh just before the removal of the vertex that caused the birth/death of *t*



By storing the ***simplification history*** (faces+errors) we can simply extract ***any approximation level*** in real time

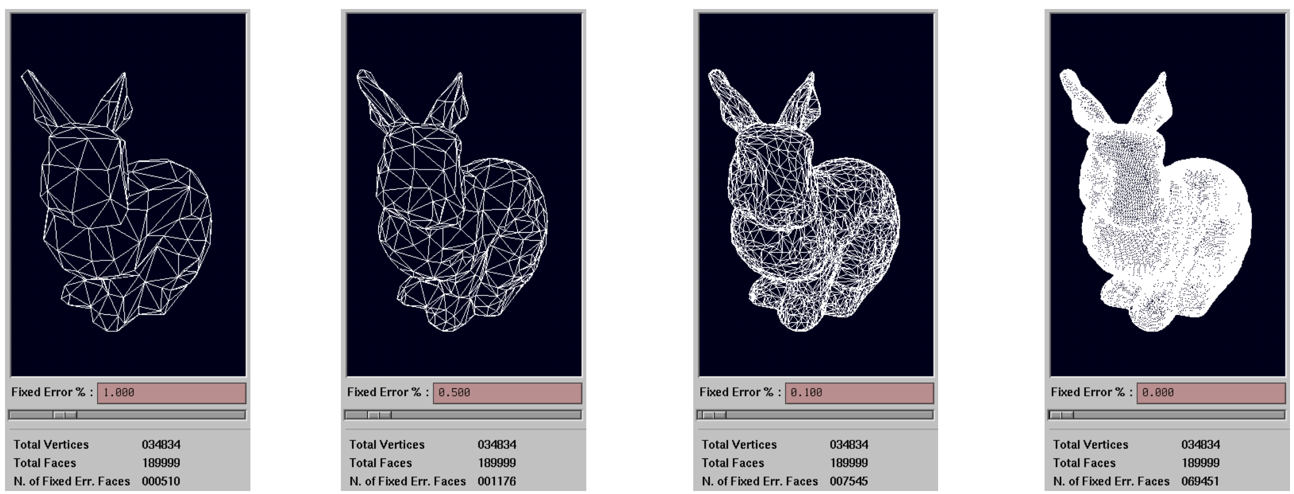
Real-time resolution management

- by extracting from the ***history*** all the triangles $\boldsymbol{t_i}$ with

  $$t_b \leq \varepsilon < t_d$$

  we obtain a model $\mathbf{M}_{\boldsymbol{\varepsilon}}$ which satisfies the approximation error $\boldsymbol{\varepsilon}$

- mantaining the whole *history* data structure costs approximately 2.5x - 3x the full resolution model

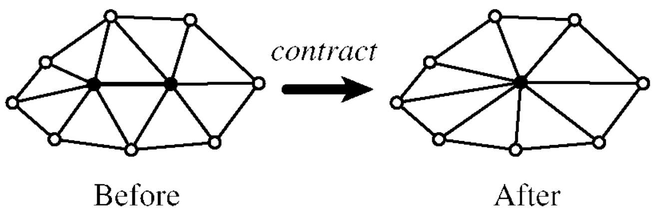


real-time LOD

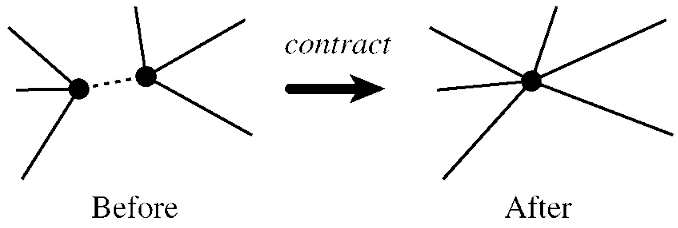
Quadric Error Metrics

Simplification using Quadric Error Metrics [Garland et al. Sig'97]

- Based on incremental **edge-collapsing**



- **but** can also collapse vertex couples which are **not connected** (topology is not preserved)



S I M P L I F I C A T I O N | A L G O R I T H M S

Geometric error approximation is managed by simplifying an approach based on **plane set distance**
[Ronfard,Rossignac96]

- INIT_time: store for each vertex the set of incident planes
- Vertex_Collapsing $(v_1, v_2) \Rightarrow v_{new}$
  - plane_set (v_{new}) = union of the two **plane sets** of v_1, v_2
  - collapse only if v_{new} is not “farther” from its plane set than the selected target error **ε**

criticism:

- storing plane sets and computing distances is not cheap !

Quadric Error Metrics solution:

- quadratic distances to planes represented with **matrices**
  - plane sets merge *via* matrix sums
  - very efficient evaluation of error *via* **matrix operations**

  but
  - triangle size is taken into account only in an approximate manner (orientation only in Quadrics + weights)

Algorithm structure:

- select valid vertex pairs (upon their distance),

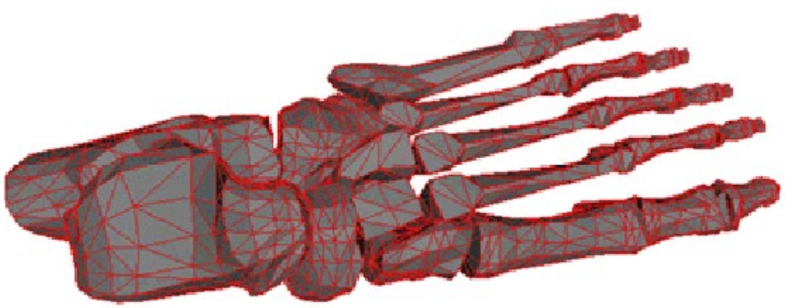
  insert them in an heap sorted upon minimum cost;
- **repeat**
  - extract a valid pair v_1, v_2 from heap and contract into v_{new};
  - re-compute the cost for all pairs which contain v_1 or v_2 and update the heap;

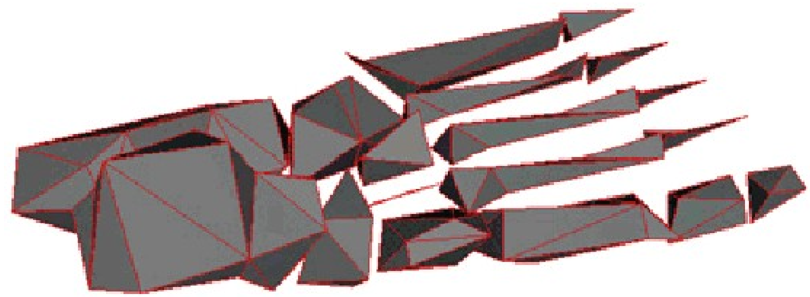
  until sufficient reduction/approximation **or** heap empty

An example

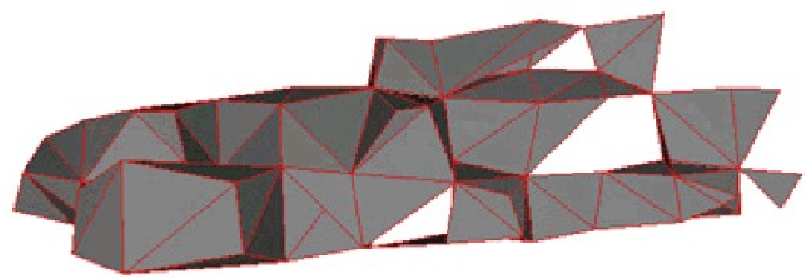
- **Original.** Bones of a human's left foot (4,204 faces).
- Note the many separate bone segments.

- **Edge Contractions.** 250 face approximation.
- Bone seg-ments at the ends of the toes have disappeared; the toes appear to be receding back into the foot.



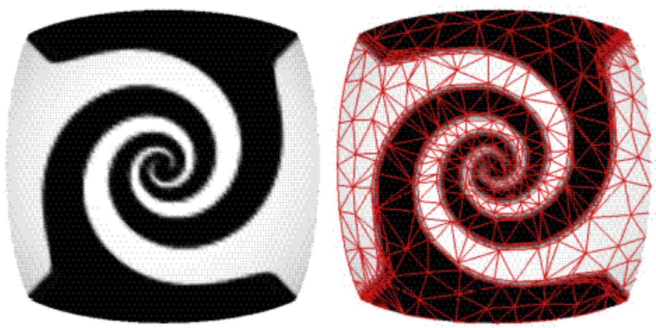


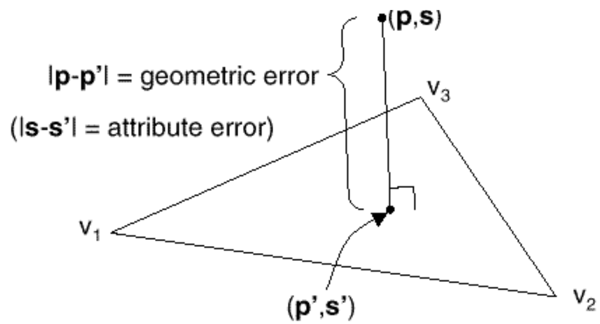
- **Clustering.** 262 face approximation.



[Images by Garland and Heckbert]

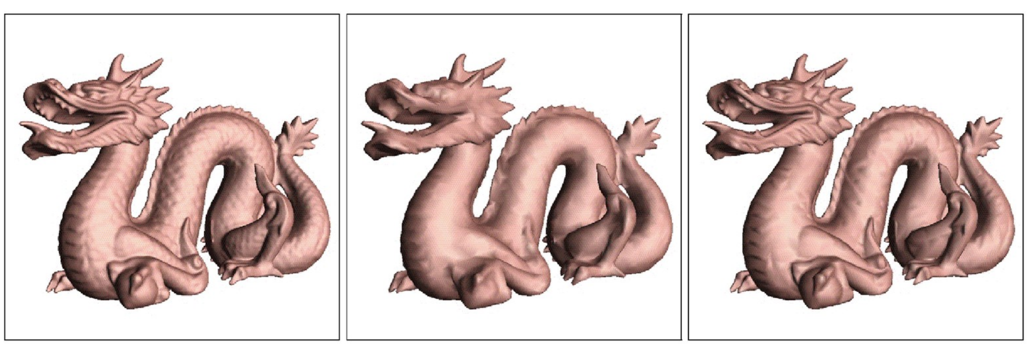
... Quadric Error Metrics Extension ...





Quadric can be exteneded to take into account:

- color and texture attributes error are computed by projecting them in R^{3+m} [Garland 98]
- by computing attribute error as the squared deviation between original value and the value interpolated [Hoppe 99]



(a) Original mesh (b) Q is just geometric error (c) Q also includes normals

Quadric Error Metrics -- *Evaluation*

- iterative, incremental method
- error is bounded
- allows topology simplification (aggregation of disconnected components)
- results are very high quality and ***times incredibly short***
- Various commercial packages use this technique (or variations)

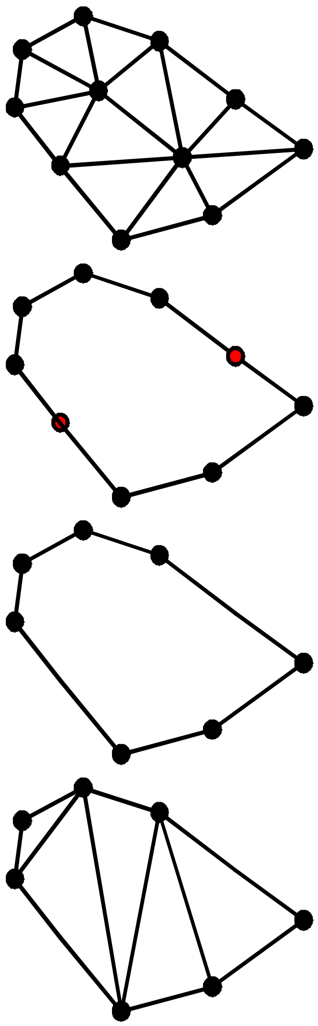
Not-incremental methods:

- coplanar facets merging [Hinker et al. `93, Kalvin et al. `96]
- re-tiling [Turk `92]
- clustering [Rossignac et al. `93, ... + others]
- wavelet-based [Eck et al. `95]

Coplanar Facets Merging

Geometric Optimization **[Hinker '93]**

- Construct nearly co-planar sets (comparing normals)
- Create edge list and remove duplicate edges
- Remove colinear vertices
- Triangulate resultant polygons



Geometric Optimization - Evaluation

- simple and efficient heuristic
- evaluation of approximation error is highly inaccurate and not bounded (error depends on relative size of merged faces)
- vertices are a subset of the original
- preserves geometric discontinuities (e.g. sharp edges) and topology

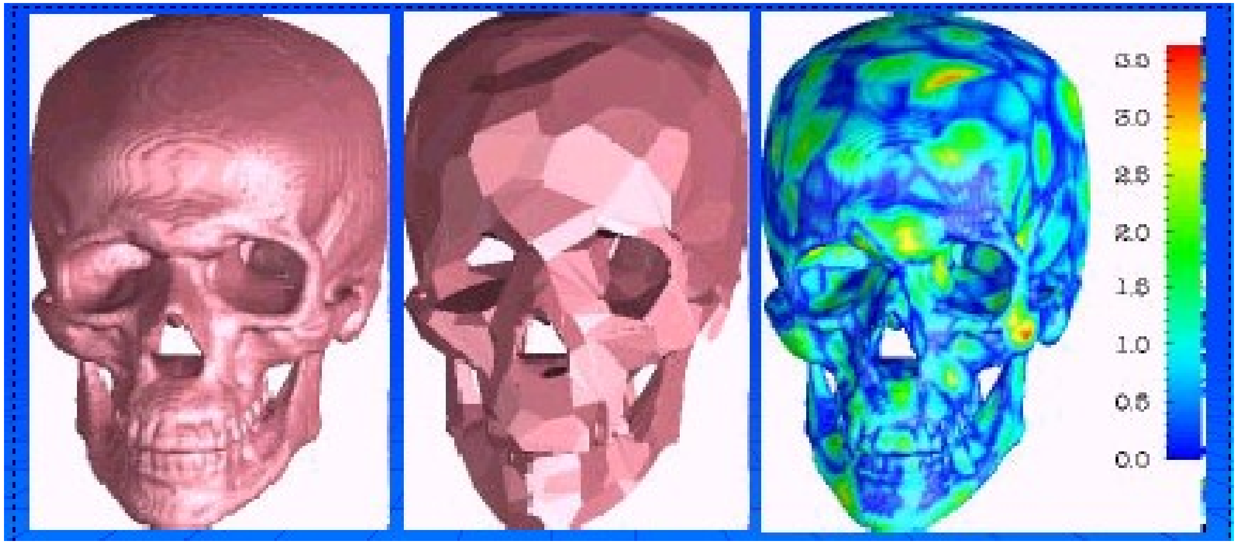
Superfaces

[Kalvin, Taylor '96]

- group mesh faces in a set of *superfaces*:
  - iteratively choose a seed face f_i as the current *superface* Sf_j
  - find by propagation all faces adjacent to f_i whose vertices are at distance $\varepsilon/2$ from the mean plane to Sf_j and insert them in Sf_j
  - moreover, to be merged each face must have orientation similar to those of others in Sf_j
- straighten the *superfaces* border
- re-triangulate each *superface*

Superfaces - an example

- Simplification of a human skull (fitted isosurface), *images courtesy of IBM*



Superfaces - Evaluation

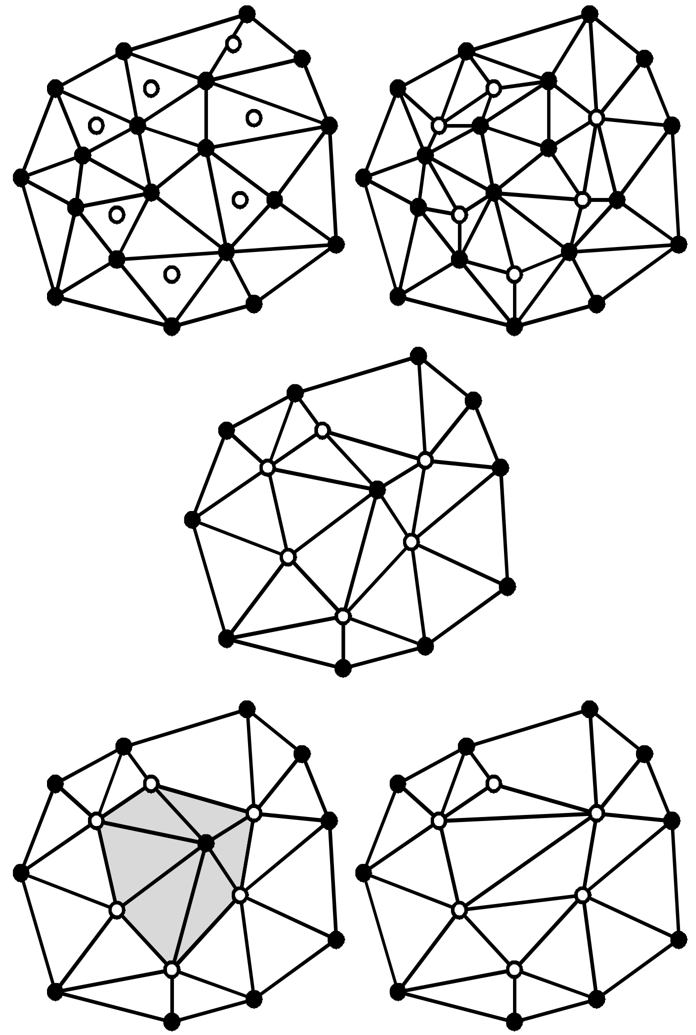
- slightly more complex heuristics
- evaluation of approximation error is more accurate and bounded
- vertices are a subset of the original ones
- preserves geometric discontinuities (e.g. sharp edges) and topology

Re-tiling

Re-Tiling **[Turk `92]**

- Distribute a new set of vertices into the original triangular mesh (points positioned using repulsion/relaxation to allow optimal surface curvature representation)
- Remove (part of) the original vertices
- Use local re-triangulation

no info in the paper on time complexity!

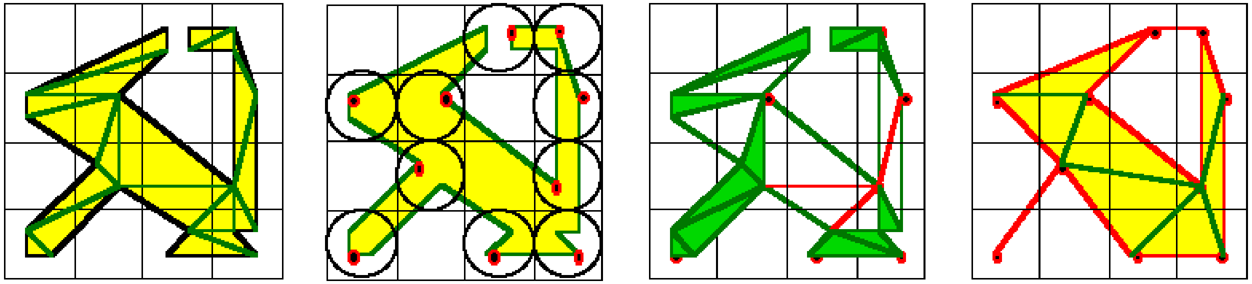


Vertex Clustering

[Rossignac, Borrel `93]

- detect and unify *clusters* of nearby vertices

  (discrete gridding and coordinates truncation)
- all faces with two or three vertices in a cluster are removed
- does not preserve topology (faces may degenerate to edges, genus may change)
- approximation depends on grid resolution



(figure by Rossignac)

Clustering -- Examples (1)

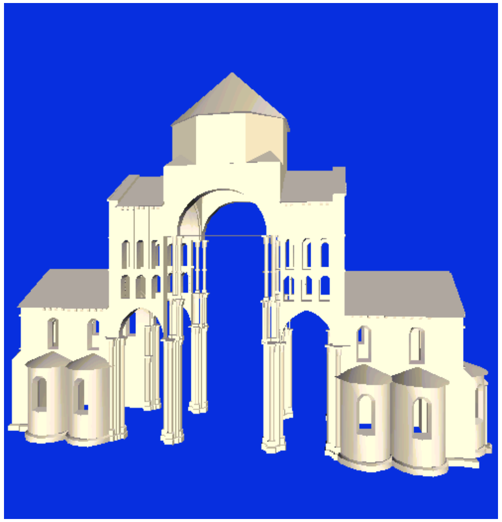
- Simplification of a table lamp, IBM 3D Interaction Accelerator, courtesy IBM

10,108 facets 1,383 facets 474 facets 46 facets

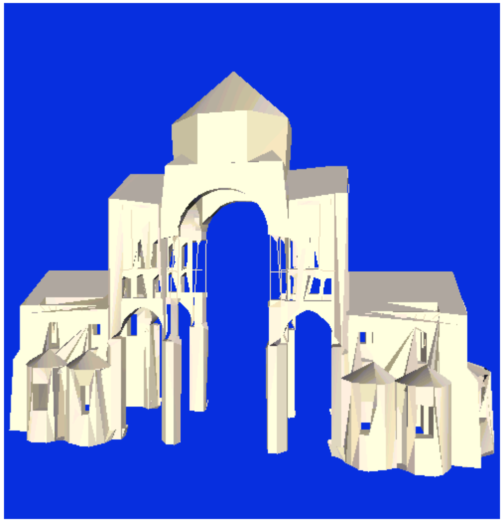
S I M P L I F I C A T I O N | A L G O R I T H M S

Clustering -- Examples (2)

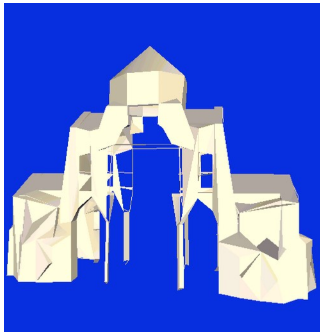
- Simplification of a portion of Cluny Abbey, IBM 3D Interaction Accelerator, courtesy IBM France.



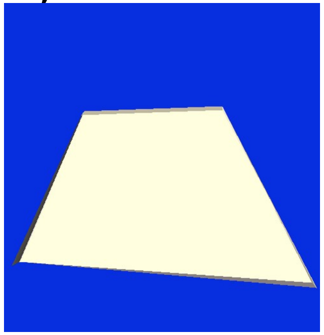
46,918 facets



6,181 facets



1,790 facets



16 facets

Clustering - *Evaluation*

- high efficiency (but timings are not reported in the paper)
- very simple implementation and use
- low quality approximations
- does not preserve topology
- error is bounded by the grid cell size

part of IBM 3D Interaction Accelerator

Multiresolution Analysis

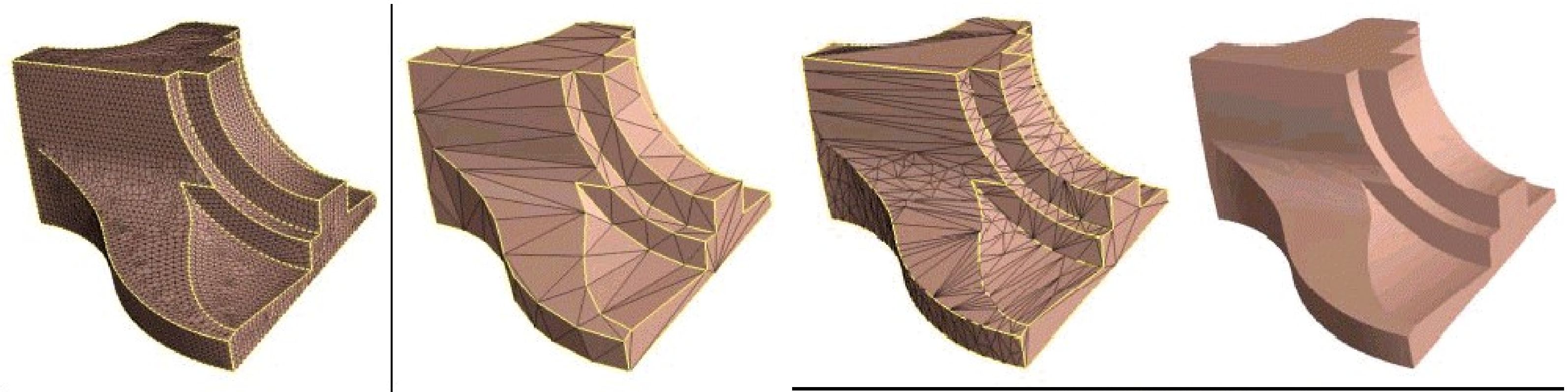
[Eck et al. '95, Lounsbery'97]

- Based on the ***wavelet*** approach
  - simple base mesh
  - + local correction terms (wavelet coefficients)
- Given input mesh M:
  - ***partition*** : build a low resolution base mesh K_0 with tolerance ε_1
  - ***parametrization*** : for each face of K_0 build a parametrization on the corresponding faces of M
  - ***resampling*** : apply ***j*** recursive quaternary subdivision on K_0 to build by parametrization different approximations $\boldsymbol{K_j}$
- Supports:

  bounded error, compact multiresolution repr., mesh editing at multiple scales

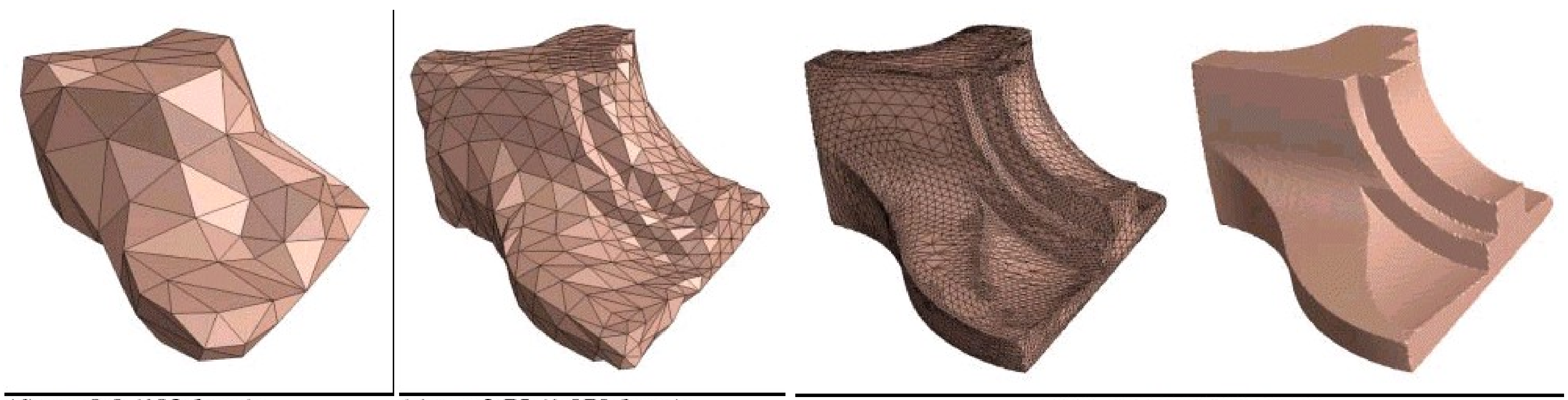
Hoppe's experiment: comparative eval. of quality of multiresolution representation

- Progressive Meshes



(a) $\hat{M}$ (12,946 faces) (b) M^{75} (200 faces) (c) M^{475} (1,000 faces)

- Multiresolution Analysis



(d) $\epsilon = 9.0$ (192 faces) (e) $\epsilon = 2.75$ (1,070 faces) (f) $\epsilon = 0.1$ (15,842 faces)

Multires Signal Processing for Meshes [Guskov, Swelden,Schroeder 99]

- Still the ***Partition, Parmetrization and Resampling*** approach but the original mesh connectivity is retained:
  - partition is done on the simplified mesh
  - use of a ***non-uniform relaxation procedure*** (instead of standard triangle quadrisection) that mimics the inverse simplification process
  - Possibility of using signal processing techniques on mesh (eg. Smoothing, detail enhancement …)

Preserving detail on simplified meshes

- Problem Statement :

  how can we preserve in a *simplified* surface
  the **detail** (or **attribute value**)
  defined on the *original* surface ??

- What one would preserve:
  - **color** (per-vertex or texture-based)
  - **small variations of shape curvature** (bumps)
  - **scalar fields**
  - **procedural textures** mapped on the mesh

Approaches proposed in literature are:

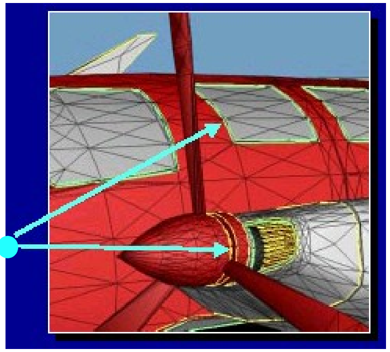
- **integrated** in the simplification process
  (ad hoc solutions **embedded** in the simplification codes)

- **independent** from the simplification process
  (post-processing phase to restore attributes detail)

Integrated approaches:

- attribute-aware simplification
  - do not simplify an element ***e*** **IF** ***e*** is on the boundary of two regions with different attribute values

  or
  - use an enhanced multi-variate approximation evaluation metrics (shape+color+…) [Hoppe96,GarHeck98,Frank etal98, Cohen etal98]



(image by H. Hoppe)

- store removed detail in textures
  - *vertex-based* [Maruka95 , Soucyetal96]
  - *texture-based* [Krisn.etal96]
- preserve **topology** of the attribute field [Bajaj et al.98]

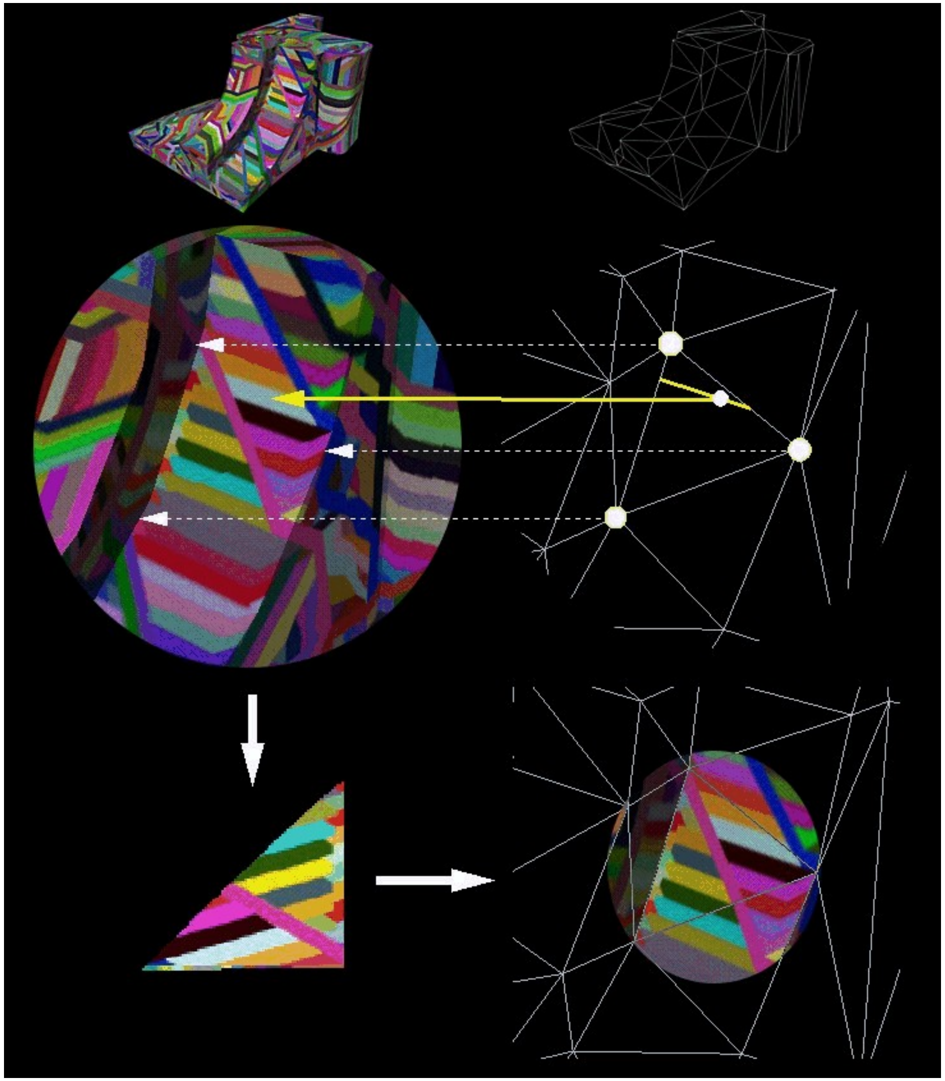
Simplification-Independent approach:

our Vis'98 paper [Cignoni etal 98]

- ***higher generality:*** attribute/detail preservation is not part of the simplification process
- performed as a ***post-processing*** phase (after simplification)
- any attribute can be preserved, by constructing an ad-hoc ***texture map***

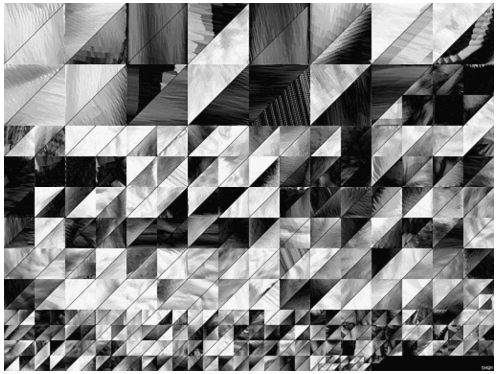
A simple idea:

- for each simplified face:
  - detect the original detail
  - code it into a triangular texture map
- pack all textures patches in a std. rectangular texture



More in detail:

- For each triangular face produce a texture patch, which encodes the "detail" of **S** lost in $\mathbf{S_1}$
  - scan-convert each face of simplified mesh $\mathbf{S_1}$
    - for each sample point **p**:
      - find the corresponding point **p'** on original **S**
      - **compute** the attribute value in **S** on **p'**
      - **store** this value in a **triangular texture patch**



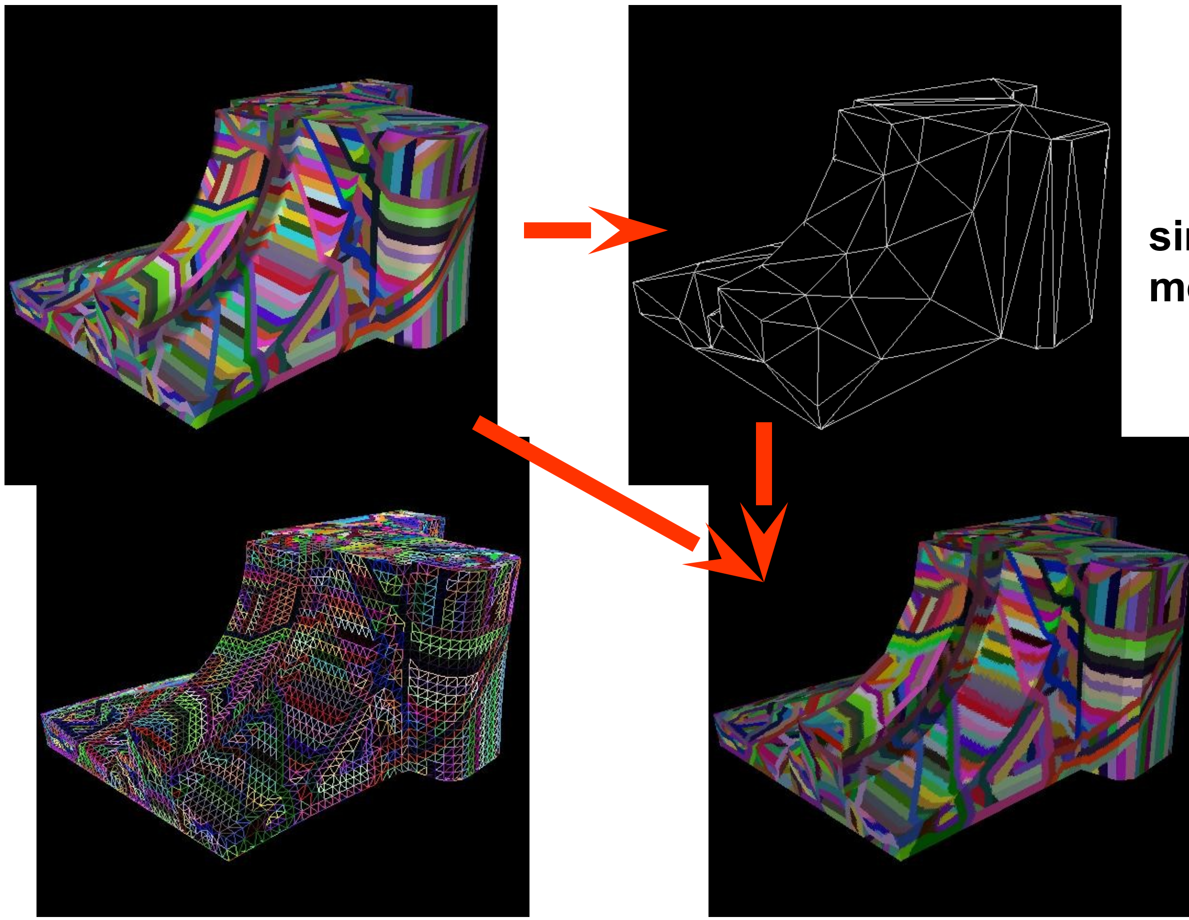
- Texture patches are stored in an efficient manner into a single, rectangular texture
- Use std. texture mapping (sw/hw) to render in real time

Times: tens of seconds

... Preserving detail: Simplif.-Independent...

- an example of ***color*** preservation

original mesh (per-vertex color)

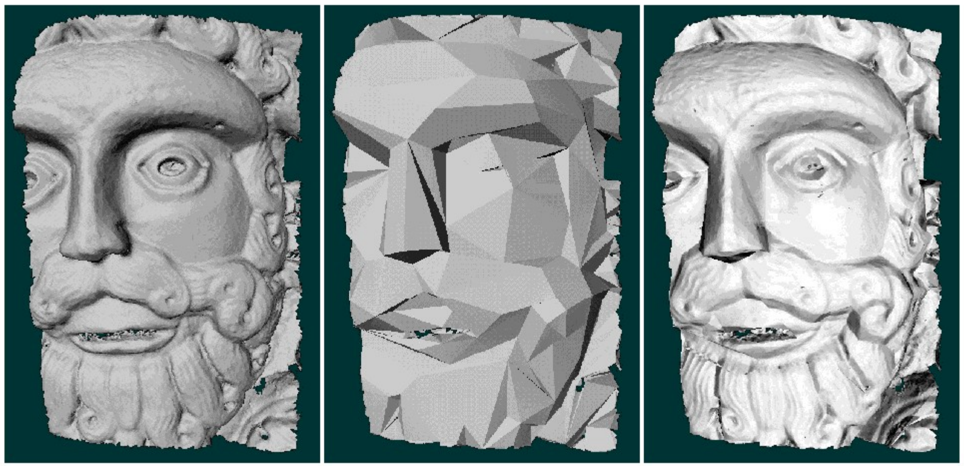


simplified mesh

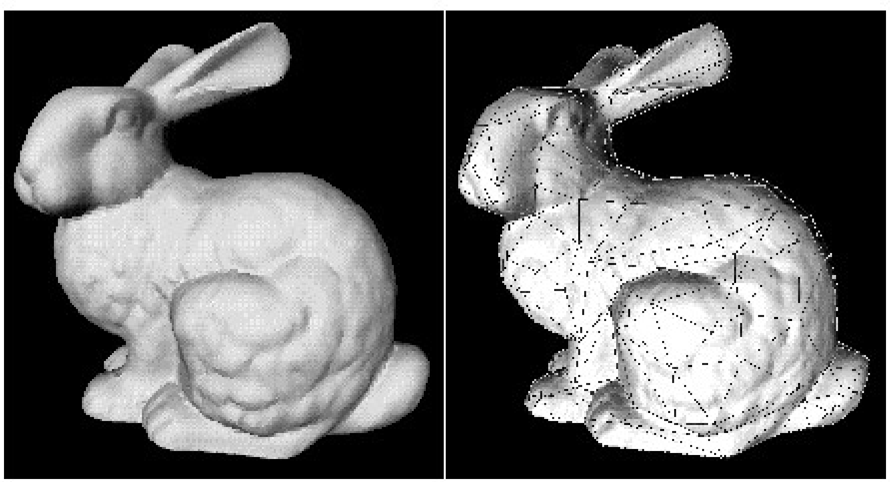
simplified mesh with textured color

... Preserving detail: Simplif.-Independent...

- example of ***geometric detail*** preservation by ***displacement mapping***



Original 20k face
simplified 500 face



Original 60k faces
simplified 250 faces