Fondamenti di Grafica Tridimensionale

Paolo Cignoni
p.cignoni@isti.cnr.it
http://vcg.isti.cnr.it/~cignoni
Simplification Algorithms

- **Simplification approaches:**
  - incremental methods based on local updates
    - mesh decimation [Schroeder et al. 92]
    - energy function optimization [Hoppe et al. 93, 96, 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:**
  - **vertex removal**
  - **edge collapse**
    - preserve location
    - new location
  - **triangle collapse**
    - preserve location
    - new location

<table>
<thead>
<tr>
<th>Action</th>
<th>Diagram</th>
<th>No. Faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex removal</td>
<td>![Vertex Removal Diagram</td>
<td>n-2</td>
</tr>
<tr>
<td>edge collapse</td>
<td>![Edge Collapse Diagram]</td>
<td>n-2</td>
</tr>
<tr>
<td>triangle collapse</td>
<td>![Triangle Collapse Diagram]</td>
<td>n-4</td>
</tr>
</tbody>
</table>
... 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
approximation quality evaluated with an **energy function**:

\[ E(M) = E_{\text{dist}}(M) + E_{\text{rep}}(M) + E_{\text{spring}}(M) \]

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

- \( E_{\text{dist}} \): sum of squared distances of the original points from \( M \)
- \( E_{\text{rep}} \): factor proportional to the no. of vertex in \( M \)
- \( E_{\text{spring}} \): sum of the edge lengths
Algorithm structure

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

- **inner minimization cycle** *(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
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
Energy function optimization: **Progressive Meshes**

**Progressive Meshes**

[Hoppe `96]

- execute **edge collapsing only** to reduce the energy function

- **edge collapsing** can be easily inverted == \(\Rightarrow\) store sequence of inverse vertex split transformations to support:
  - multiresolution
  - progressive transmission
  - selective refinements
  - geomorphs

- faster than MeshOptim.
Preserving mesh appearance

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

Managed by inserting two new components in the energy function:

- $E_{\text{scalar}}$: measures the accuracy of scalar attributes
- $E_{\text{disc}}$: measure the geometric accuracy of discontinuity curves
Progressive Meshes

Examples

(a) Base mesh $M^0$ (150 faces)  
(b) Mesh $M^{175}$ (500 faces)  
(c) Mesh $M^{175}$ (1,000 faces)  
(d) Original $M = M^n$ (13,546 faces)
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.

*An implementation is present as part of DirectX 6.0 tools*
**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 $\varepsilon_{\text{max}}$

- **boundary / interior / corner vertices**:
  - if distance *vertex - new boundary/interior edge* is lower than $\varepsilon_{\text{max}}$

- adopts *local evaluation* of the approximation!!

- $\varepsilon_{\text{max}}$ : value selected by the user
Re-triangulation

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

control *aspect ratio* to ensure simplified mesh quality

- for each vertex removed:
  - *if* simple or boundary vertex $\implies$ 1 loop
  - *if* interior edge vertex $\implies$ 2 loops
  - *if* boundary vertex $\implies$ -1 face
  - otherwise $\implies$ -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)
Original Mesh Decimation - *Evaluation*

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

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

- **locally-bounded** error, based on mesh distances
  [ex. standard Mesh Decimation]

- **globally bounded** error, based on mesh distances
  [ex. Envelopes + enhanced Decimation + others]

- control based on **mesh characteristics**
  [ex. vertex proximity, mesh curvature]

- **energy function** evaluation
  [ex. Mesh Optim., Progr. Meshes]

**User’ viewpoint:**
- simple to grasp
- simple to drive
  
  very handy

  may be misleading

  not easy, many parameters to be selected
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)
Heuristics proposed for *global error evaluation*:

- **input mesh -- to -- simplified mesh edges distance**
  - [Ciampalini97]
  - for each internal edge:
    - select sampling points $p_i$ (regularly/random)
    - evaluate distance $d(M_0, p_i)$
  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

- given the input mesh $M$

  - build two envelope meshes $M_-$ and $M_+$ at distance $-\zeta$ and $+\zeta$ 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** !!

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

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

(drawn by A. Varshney)
... Enhancing Decimation - **Simplification Envelopes** ...

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*
Results

- Simplification times \(\sim\) linear with mesh size

![Graph showing Stanford Bunny Reduction - Error Graph with no staircase abrupt error increase](image)

*no staircase abrupt error increase (fundamental for the quality of the multiresolution output)*
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 $t_i$ with $t_b \leq \varepsilon < t_d$
  
  we obtain a model $M_{\varepsilon}$ which satisfies the approximation error $\varepsilon$

- maintaining the whole history data structure costs approximately $2.5x - 3x$ the full resolution model
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)
Geometric error approximation is managed by simplifying an approach based on **plane set distance** [Ronfard,Rossignac96]

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

**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 segments 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]
Let \( \mathbf{n}^T \mathbf{v} + d = 0 \) be the equation representing a plane.

The squared distance of a point \( \mathbf{x} \) from the plane is

\[
D(\mathbf{x}) = \mathbf{x}(\mathbf{n} \mathbf{n}^T)\mathbf{x} + 2d\mathbf{n}^T\mathbf{x} + d^2
\]

This distance can be represented as a quadric

\[
Q = (A, \mathbf{b}, c) = (\mathbf{n} \mathbf{n}^T, d\mathbf{n}, d^2)
\]

\[
Q(\mathbf{x}) = \mathbf{x}A\mathbf{x} + 2\mathbf{b}^T\mathbf{x} + c
\]
The boundary error is estimated by providing for each boundary vertex \( v \) a quadric \( Q_v \) representing the sum of all the squared distances from the faces incident in \( v \).

- The error of collapsing an edge \( e= (v, w) \) can be evaluated as \( Q_w(v) \).
- After the collapse the quadric of \( v \) is updated as follow \( Q_v = Q_v + Q_w \).
Error

Domain Error

- The two dataset D and D’ span different domains \( \Omega, \Omega' \)
- Same problem of classical surface simplification
- Measure the Hausdorff distance between the boundary surfaces of the two datasets D and D’
  \[ e^a_f(D, D') = \max \left( \min(\text{dist}(x,y)) \right) \]
  \[ x \in \Omega \quad y \in \Omega' \]
  \[ e_f(D, D') = \max(e^a_f(D, D'), e^a_f(D', D)) \]

- Various techniques to approximate this distance between two surfaces [Ciampalini et al. 97]
Quadric can be extended to take into account:

- color and texture attributes error are computed by projecting them in $\mathbb{R}^{3+m}$ [Garland 98]

- by computing attribute error as the squared deviation between original value and the value interpolated [Hoppe 99]
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]
**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
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 $\epsilon/2$ 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

[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
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
Wavelet methods

Multiresolution Analysis

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 $\varepsilon_1$
- **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 $K_j$

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) $M^{15}$ (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, Parametrization 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 ...)

*image by courtesy of Guskov et al. 99*
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+...)  

- store removed detail in textures
  - *vertex-based*  [Maruka95, Soucy et al 96]
  - *texture-based*  [Krisn. et al 96]

- preserve **topology** of the attribute field  
  [Bajaj et al. 98]
... Preserving detail: Simplif.-Independent...

Simplification-Independent approach:

**our Vis’98 paper** [Cignoni et al. 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**

- Used today in most games...
A simple idea: ... Preserving detail: Simplif.-Independent...

- for each texel simplified face:
  - detect the original detail by choosing either the closest point or along the normal.
... 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 normal mapping:

- Original 20k faces simplified to 500 faces.
- Original 60k faces simplified to 250 faces.