Mesh Simplification: Algorithms and Strategies

Richard Zhang

CMPT 464/764: Geometric Modeling in Computer Graphics
Lecture 8
What to simplify

- **Geometry**
  - Primitives to remove: vertices, edges, faces, etc.
  - How to fix connectivity
  - How to determine geometric positions

- **Topology**
  - Global: *disconnected parts, holes* — simplification may join them
  - Local: *manifold* vs. *non-manifold* vertices and edges
Algorithm classification

- **Refinement-based vs. decimation-based**
  - Refinement = top-down = coarse-to-fine: e.g., subdivision, Douglas-Peuker
Algorithm classification

- **Refinement-based vs. decimation-based**
  - Refinement = top-down = coarse-to-fine: e.g., subdivision, Douglas-Peuker
Algorithm classification

- **Refinement-based** vs. **decimation-based**
  - Refinement = top-down = coarse-to-fine: e.g., subdivision, Douglas-Peuker
Algorithm classification

- **Refinement-based** vs. **decimation-based**
  
  - Refinement = top-down = coarse-to-fine: e.g., subdivision, Douglas-Peuker
Algorithm classification

- **Refinement-based vs. decimation-based**
  - Refinement = top-down = coarse-to-fine: e.g., subdivision, Douglas-Peuker
  - Decimation-based = bottom-up = fine-to-coarse: most common for meshes with irregular connectivity (unstructured)
Algorithm classification

- **Local vs. global**
  - Most algorithms are local, through local geometry processing
  - An example of global approach with error bound: use of simplification envelopes

- **Fidelity-based vs. budget-based**
  - What is the constraint and what is to be optimized

- **Topology-preserving vs. topology-modifying**
Fidelity-based

- User specifies minimum fidelity criterion or maximum error tolerance
- Algorithm finds simplification with the smallest triangle count
- What fidelity measure to use? — error metrics from last lecture
  - **Object-space**: view-independent
  - **Image-space**: view-dependent but this is what matters
  - **Perceptual concerns**: not yet fully understood and only a few simple ones implemented for LOD in graphics
Budget-based

- User specifies polygon budget, e.g., due to
  - bandwidth limitations of rendering pipeline
  - resources tied to other operations, e.g., gaming AI
- Algorithm gives best approximation based on budget
- Most frequently used paradigm
Topology-preserving

- Simplification of mesh geometry only – more faithful
- Drastic simplification limited if object genus is high

Courtesy Stanford 3D Scanning Repository
Topology-simplifying

- Simplify mesh topology by filling holes or joining disconnected parts.
Topology-preserving vs. T-modifying

- **Topology-preserving simplification**
  - Better visual fidelity — less change to the mesh
  - Smoother transitions between levels
  - Limits drastic simplification when topology is complex

- **Topology-modifying simplification: the opposite**
  - Can have more drastic simplification – e.g., fill holes
  - Can deal with non-manifold mesh better
  - Poorer visual fidelity and popping when filling a hole
Global operators for topology

- When input scan is noisy and reconstruction algorithm is imperfect
  - Non-manifold vertices or edges — easy to detect but hard to resolve
  - False topological features, e.g., self-intersections or unwanted holes — not easy to detect

- Topological features, e.g., a hole, are of a more global nature — hard to detect locally — hence need to use global operators

- Topology simplification is often more complex
One approach: $\alpha$-hulls

- **Intuition:**
  - Imagine a point set embedded in a space of styrofoam
  - Use a spherical eraser of radius $\alpha$ to erase as much foam as possible while **constrained by the set of points**
  - Remaining foam forms the $\alpha$-hull

- **Special cases:**
  - $\alpha = 0$: $\alpha$-hull is the point set itself
  - $\alpha = \infty$: $\alpha$-hull = convex hull
\( \alpha \)-hull to simplify topology

- Goal: remove complex topological noise from mesh
- Imagine:
  - Hard mesh surface is embedded in a space of styrofoam
  - Roll a sphere of radius \( \alpha \) over mesh surface
- Resulting foam (the \( \alpha \)-hull) forms final mesh
  - All spaces of foam not reachable by sphere are retained
  - Thus holes get filled if the sphere cannot “peek in”
α-hull to simplify topology

- Rolling sphere over complex surface is hard to simulate
  - Equivalent: **roll a point** over the mesh surface that is **grown** by α
- Surface growing: **Minkowski sum** or **dilation** (math morphology)
  - Erosion (shrinkage) follows dilation (growth) in math morphology

- Computing Minkowski sum, even for a sphere, is hard on a mesh
\(\alpha\)-hull to simplify topology (aside)

- In practice, **grow with a cube** (e.g., axis-aligned cube)
  - Simpler growing operation, e.g., only polyhedra to deal with
  - Decompose mesh into triangles, grow, and compute union
  - Growing cubes from triangles results in **convex** polyhedra
  - Linear algorithm exists to union convex polyhedra
Our focus: local decimation

- Remove one vertex/face/edge or combinations of these at a time and fix up mesh afterwards --- topology simplification possible
- Reduce mesh complexity by a small amount at each step
- Most algorithms use one simplification operator, e.g., edge collapse
- Key: find right order to simplify = outer optimization
Op. 1: Edge collapse

- Half-edge collapse: just move to one of the end point
- Full-edge collapse: new location via **continuous optimization**
- May create **mesh fold-over** – how to detect?
- May create **non-manifold triangles** – how to detect?
Artifacts by edge collapse

Detect by looking at change of face normal

Non-manifold edge! Do not collapse
Op. 2: Vertex pair collapse

- Similar to edge collapse
- Goal: allow nonadjacent but nearby vertices to merge – can join different parts of a mesh to lead to **topology changes**
- Treated as a **virtual edge** collapse
Op. 3: Triangle collapse

- Collapse a triangle into one vertex $P$
- Position of $P$ can be at $H$, $F$, $G$, or newly computed point
- One triangle collapse = two edge collapse, e.g., $HG$ then $FG$

Triangle Collapse

[Hamman 94]

Point Split

Triangle Collapse
Op. 4: Vertex removal

- Hole left by vertex removal is triangulated
- One way to retriangulate is equivalent to a half-edge collapse
- Discrete optimization to find optimal re-triangulation

[Schroeder 92]
Op. 5: Cell collapse

- All vertices in a cell (e.g., grid cell or cell in an octree) are collapsed to a single point – as in **graph contraction**
- Single point in cell can be an old vertex or an average
- Aggressive geometry- and **topology-modifying** decimation
- Result **not invariant** under rotation or translation due to grid change
Simplification strategies

- An **optimization process** to ensure approximation quality
- There are **two optimizations**: 
  - **Outer optimization**: in which order should the primitives be decimated
  - **Inner optimization**: how to choose new vertex position or triangulation
  - Both depends on simplification error metric
- Finding global optimum is hard, so …?
Strategy 1: Non-optimizing

- Example: cell collapse

- **No outer optimization**
  - Cell collapse does not care about order of cells

- Inner optimization may still be performed instead of just computing some simple average
Strategy 2: Greedy algorithms

- Idea: find the simplification operation, e.g., an edge collapse, that induces the least error at current stage
- Place eligible edge collapses in priority queue (heap)
- Collapse the head of heap and update (delete & insert) edge collapse operations affected
- For efficiency:
  - Simplify error estimate, e.g., instead of Quadric, use edge length
  - Reduce # of edges to re-compute (those needing re-heap), e.g., lazy computation
Strategy 3: Lazy computations

- **Observation:** an edge collapse may be re-evaluated (i.e., its cost computed) many times before being collapsed (as top of heap)

- **Idea of lazy computations:**
  - I know this edge (cost) is to be recomputed but I am **lazy**
  - Leave it there but mark it as **dirty** – hope not too far off the actual cost
  - **Compute actual cost only when it becomes a candidate,** e.g., at the top of the heap, and then re-insert into heap
Details on lazy computations

1. For each candidate operation op
2. ComputeCost(op);
3. op.dirty = false;
4. Q.insert(op);
5. While Q is not empty
6.   op = Q.extractMin();
7.   if (!op.dirty)
8.     ApplyOp(op);
9.   for i = neighbor op
10.       i.dirty = true;
11. else
12.   ComputeCost(op);
13.   op.dirty = false;
14.   Q.insert(op);

- Dirty op with lower actual cost may be “overlooked”

Experimental result:
- Similar simplification error between greedy and lazy approaches
- But significantly less cost computations

[Cohen 97]
Strategy 4: Multiple-Choice Algorithm

- One type of **randomized** optimization technique
- As an alternative to the greedy approach for efficiency
- Instead of choosing the best based on a priority queue, choose the **best among a small set of random choices**
- Faster than greedy with comparable decimation quality
- For more details, see paper by [Wu and Kobbelt 2005] – assignment #2 programming problem
Most notable decimation algorithms

- Vertex clustering [Rossignac 92]
- Vertex decimation + re-triangulation [Schroder et al. 92]
- Progressive meshes based on edge collapse [Hoppe 96]
- Quadric-based vertex pair collapse [Garland & Heckbert 97]
- Refinement-based algorithms – RSIMP = reverse simplification (See LOD Book) [Brodsky and Watson 00]
- Image-driven approach – really expensive [Lindstrom 00]
Algorithm 1: Vertex clustering

- Fast, robust, simple to implement, e.g., [Rossignac 92]

**Overview**
- Assign an **importance** (adjacent to large faces and/or represent features, e.g., high curvatures or silhouette) to each vertex
- Create grid and collapse all vertices in a grid cell to the most important vertex or an importance-weighted average
- Filter out degenerate triangles and edges
- Resolution of grid dictates simplification quality
Algorithm 2: Vertex decimation

- Vertex removal + re-triangulation [Schroeder et al. 92]
- **Multiple passes**: decimate *as long as* error is below given threshold
- Vertices are characterized as
  1. Simple: error is distance to an *average plane*
  2. Interior edge: error is distance to an *average line*
  3. Corner: do not remove if believed to be feature
  4. Non-manifold: do not remove
  5. Boundary: error is distance to an *average line*
Average plane and line

- How to compute average plane for a simple vertex:
  - Use one ring of triangles surrounding the vertex
  - Normal of plane is area-weighted face normals
  - A point on the plane is area-weighted centroids of the triangles

- Average line:
  - Formed by two vertices on boundary/feature edges
  - If error < threshold, then decimate
Re-triangulation

- Total number of possibly ways to triangulate an $n$-sided convex polygon: $(2n-4)! / [(n-1)!(n-2)!]$ – the Catalan sequence
- Choose one which minimizes an approximation error is a discrete approximation problem
- Heuristics (e.g., ring heuristic) or more sophisticated optimization
Algorithm 3: Progressive mesh (PM)

- One of the most well-known [Hoppe 96]
- PM stores a mesh as a base mesh plus a sequence of vertex splits
- PM constructed via edge collapses directed by a greedy algorithm
- Greedy algorithm finds an edge collapse that introduces the smallest change to an energy functional
Inner optimization

- Given a chosen edge collapse operation

\[(u, w) \rightarrow v\]

- Optimize the position of \(v\) so that the \textit{geometric} error:

  \textit{sum of squared distances from} \(u\) \textit{and} \(w\) \textit{to their respective closest simplified triangle}

  \textit{is minimized}

- Problem: the optimal position of \(v\) may be at infinity!
Optimal location at infinity
Mesh optimization

- To fix it, enforce a **spring energy term**, e.g., penalizes long edges.
- Outer optimization is based on minimizing changes to the functional

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

- \(E_{\text{dist}}\): geometric error
- \(E_{\text{spring}}\): spring energy
- \(E_{\text{scalar}}\): color and other scalar attributes
- \(E_{\text{disc}}\): penalty for altering discontinuity curves, e.g., sharp creases
Algorithm 4: Quadric-based

- **Most frequently adopted** mesh decimation algorithm
- Vertex-pair collapse based on error quadrics [Garland & Heckbert 97]
  - Combines speed, robustness, and fidelity – *(Qslim, MeshLab)*
  - Iteratively **merges close vertex pairs** until desired level of decimation reached – topology-modifying
  - Candidate vertex pairs placed in a heap based on **quadric errors**
A quadric is a $4 \times 4$ symmetric matrix capturing information about one or more planes.

Given a plane $p$ specified by $p = [a \ b \ c \ d]^T$, its fundamental error quadric is given by

$$K_p = pp^T = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

Only needs 10 floats to store – memory-efficient
Point to plane distance

\[
K_p = pp^T = \begin{bmatrix}
a^2 & ab & ac & ad \\
ab & b^2 & bc & bd \\
ac & bc & c^2 & cd \\
ad & bd & cd & d^2
\end{bmatrix}
\]

- Plane equation is \( p^Tw = 0 \), where \( w = [x \ y \ z \ 1]^T \)
- **Squared distance from a point** \( v \) **to plane** \( p \) **is** \( v^Tpp^Tv \):
  
  Assume that \([a \ b \ c]\), the plane normal, is of unit length
  
  Intersect line \( w = v + t [a \ b \ c \ 0]^T \) with plane: \( p^Tw = 0 \implies t = -p^Tv \)
  
  The squared distance is \( t^2 = p^Tv p^Tv = v^Tpp^Tv \)
Quadric error

- Given a vertex \( v \) and a quadric \( Q \), \( v'Qv \) gives the sum of squared distances from \( v \) to planes captured by \( Q \).

- **MOST IMPORTANT:** quadrics are additive, i.e., quadric of the “union” of two sets of planes = sum of quadrics of each set.

- **Double counting** with vertex merging as they share two planes – ignored for speed considerations.

- Store one quadric at each vertex \( v \), capturing all the planes that had ever been associated with \( v \).
  - At start, the quadric for each vertex is the sum of quadrics of the vertex’s adjacent triangles.
Inner optimization

- For each pair \((u, w)\) having quadrics \(Q_u\) and \(Q_w\), find merged position \(v\) so that \(v'(Q_u + Q_w)v\) is minimized.

- \(v\) is found analytically by solving a linear system.

- Resulting quadric errors are used to form the priority queue.
Outer optimization

- Choose the highest-priority (least quadric error) pair to collapse
- Compute optimal location of merged vertex
- Update cost for all vertex pairs involving the new vertex
  - Remove head of priority queue
  - Re-compute new costs
  - Insert new pairs into priority queue
  - Repeat until desired level of decimation reached
- Lazy computations possible here