Mesh Representation and Decimation

Jason Lawrence
CS451: Advanced Computer Graphics
Spring 2008
Outline

- Mesh data structures
- Mesh decimation
Topology Basics

- Genus
- Closed vs. Open
- 2-Manifold
Topology Basics

- Genus
- Closed vs. Open
- 2-Manifold
Topology Basics

- Genus
- Closed vs. Open
- 2-Manifold

Local areas topologically equivalent to a disc
Triangle Mesh

- Explicit representation of an object’s surface
- One type of boundary representation ("B-rep")
- Why triangles?
- Piecewise linear approximation:
  - Vertices
  - Edges
  - Faces
Mesh Data Structures

- What are desirable characteristics?

- Generality (from most general to least general):
  - triangle “soup”
  - 2-manifold $\Rightarrow$ (≤ 2 triangles per edge)
  - orientable $\Rightarrow$ (consistent CW / CCW winding)
  - closed $\Rightarrow$ (no boundary)

- Compact storage
Mesh Data Structures

- What are desirable characteristics?
- Efficient support for common operations:
  - rendering
  - smoothing
  - compression / decimation
  - manipulation
  - simulations
Mesh Data Structures

- What are desirable characteristics?
- Efficient support for basic operations:
  - Given face, find its vertices
  - Given vertex, find faces touching it
  - Given face, find neighboring faces
  - Given vertex, find neighboring vertices
  - Given edge, find vertices and faces it touches
Mesh Data Structures

- Independent faces
- Indexed face set
- Adjacency lists
- Winged-edge
- Half-edge
Independent Faces

- Faces list vertex coordinates
  - redundant vertices
  - no topology information

Face Table

- $F_0$: $(x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2)$
- $F_1$: $(x_3, y_3, z_3), (x_4, y_4, z_4), (x_5, y_5, z_5)$
- $F_2$: $(x_6, y_6, z_6), (x_7, y_7, z_7), (x_8, y_8, z_8)$
Indexed Face Set

- Faces list vertex references (i.e., "pointers")

**Vertex Table**
- $v_0: (x_0, y_0, z_0)$
- $v_1: (x_1, y_1, z_1)$
- $v_2: (x_2, y_2, z_2)$
- $v_3: (x_3, y_3, z_3)$
- $v_4: (x_4, y_4, z_4)$

**Face Table**
- $F_0: 0, 1, 2$
- $F_1: 1, 4, 2$
- $F_2: 1, 3, 4$

Note CCW ordering
Indexed Face Set

- Storage efficiency?
- Which operations are supported in $O(1)$ time?

**Vertex Table**
- $v_0: (x_0, y_0, z_0)$
- $v_1: (x_1, y_1, z_1)$
- $v_2: (x_2, y_2, z_2)$
- $v_3: (x_3, y_3, z_3)$
- $v_4: (x_4, y_4, z_4)$

**Face Table**
- $F_0: 0, 1, 2$
- $F_1: 1, 4, 2$
- $F_2: 1, 3, 4$

Note CCW ordering
Full Adjacency Lists

- Store all vertex, face, edge adjacencies

**Edge Adjacency Table**
- $e_0$: $v_0, v_1$; $F_0, \emptyset$; $\emptyset, e_2, e_1, \emptyset$
- $e_1$: $v_1, v_2$; $F_0, F_1$; $e_5, e_0, e_2, e_6$

**Face Adjacency Table**
- $F_0$: $v_0, v_1, v_2$; $F_1, \emptyset, \emptyset$; $e_0, e_2, e_0$
- $F_1$: $v_1, v_4, v_2$; $\emptyset, F_0, F_2$; $e_6, e_1, e_5$
- $F_2$: $v_1, v_3, v_4$; $\emptyset, F_1, \emptyset$; $e_4, e_5, e_3$

**Vertex Adjacency Table**
- $v_0$: $v_1, v_2$; $F_0$; $e_0, e_2$
- $v_1$: $v_3, v_4, v_2, v_0$; $F_2, F_1, F_0$; $e_3, e_5, e_1, e_0$
Partial Adjacency Lists

- Store some adjacencies, **derive** others
- Many possibilities...

**Edge Adjacency Table**
- $e_0$: $v_0, v_1$; $F_0, \emptyset$; $\emptyset, e_2, e_1, \emptyset$
- $e_1$: $v_1, v_2$; $F_0, F_1$; $e_5, e_0, e_2, e_6$
- ...

**Face Adjacency Table**
- $F_0$: $v_0, v_1, v_2$; $F_1, \emptyset, \emptyset$; $e_0, e_2, e_0$
- $F_1$: $v_1, v_4, v_2$; $\emptyset, F_0, F_2$; $e_6, e_1, e_5$
- $F_2$: $v_1, v_3, v_4$; $\emptyset, F_1, \emptyset$; $e_4, e_5, e_3$

**Vertex Adjacency Table**
- $v_0$: $v_1, v_2$; $F_0$; $e_0, e_2$
- $v_1$: $v_3, v_4, v_2, v_0$; $F_2, F_1, F_0$; $e_3, e_5, e_1, e_0$
- ...

Winged Edge

- Most data stored at edge
- Vertices, faces point to one edge each

**Edge Adjacency Table**
- $e_0$: $v_0, v_1$; $F_0$, $\emptyset$; $\emptyset, e_2, e_1, \emptyset$
- $e_1$: $v_1, v_2$; $F_0, F_1$; $e_5, e_0, e_2, e_6$
- $\vdots$

**Face Adjacency Table**
- $F_0$: $v_0, v_1, v_2$; $F_1$, $\emptyset$, $\emptyset$; $e_0, e_2, e_0$
- $F_1$: $v_1, v_4, v_2$; $\emptyset, F_0, F_2$; $e_6, e_1, e_5$
- $F_2$: $v_1, v_3, v_4$; $\emptyset, F_1, \emptyset$; $e_4, e_5, e_3$

**Vertex Adjacency Table**
- $v_0$: $v_1, v_2$; $F_0$; $e_0, e_2$
- $v_1$: $v_3, v_4, v_2, v_0$; $F_2, F_1, F_0$; $e_3, e_5, e_1, e_0$
- $\vdots$
Winged Edge

- Each edge stores 2 vertices, 2 faces, 4 edges (fixed size!)
- Enough information to completely “walk around” faces or vertices
Half Edge

- Instead of single edge, 2 directed “half edges”
- Makes some operations much more efficient
Efficient Algorithm Design

- Can sometimes design algorithms to compensate for operations not supported by data structures
- Example: per-vertex normals
  - Average normal of faces touching each vertex
  - With indexed face set, vertex $\rightarrow$ face in $O(n)$
  - Naive algorithm for all vertices: $O(n^2)$
  - Can you think of an $O(n)$ algorithm
Case Study: Decimation

Division, Viewpoint, Cohen

Triangles:
41,855
27,970
20,922
12,939
8,385
4,766
Mesh Decimation

- Reduce number of polygons
  - Less storage
  - Faster rendering
  - Simpler manipulation

- Desirable properties
  - Generality
  - Efficiency
  - Produces “good” approximation

Michelangelo’s St. Matthew
Original model: ~400M polygons
Mesh Decimation

LOD #Faces
1.000 13546

Hoppe
Mesh Decimation

- Typical: greedy algorithm
  - Measure error of possible "simple" operations
  - Place operations in queue according to error
  - Perform operations in queue successively
  - After each operation, re-evaluate error metrics
Primitive Operations

- Simplify model a bit at a time by removing a few faces
  - Repeated to simplify whole mesh

- Type of operations
  - Vertex cluster
  - Vertex remove
  - Edge collapse
**Vertex Cluster**

- **Method**
  - Merge vertices based on proximity
  - Triangles with repeated vertices can collapse to edges or points

- **Properties**
  - General and robust
  - Can be unattractive if results in topology change
Vertex Remove

- **Method**
  - Remove vertex and adjacent faces
  - Fill hole with new triangles (reduction of 2)

- **Properties**
  - Requires manifold surface, preserves topology
  - Typically more attractive
  - Filling hole well not always easy
Edge Collapse

**Method**
- Merge two edge vertices to one
- Delete degenerate triangles

**Properties**
- Special case of vertex cluster
- Allows smooth transition
- Can change topology
Operation Considerations

- **Topology considerations**
  - Attention to topology promotes better appearance
  - Allowing non-manifolds increases robustness and ability to simplify

- **Operation considerations**
  - Collapse-type operations allow smooth transitions
  - Vertex remove affects smaller portion of mesh than edge collapse
Geometric Error Metrics

- Motivation
  - Promote accurate 3D shape preservation
  - Preserve screen-space silhouettes and pixel coverage

- Types
  - Vertex - Vertex distance
  - Vertex - Plane distance
  - Point - Surface distance
  - Surface - Surface distance
Vertex-Vertex Distance

- $E = \max(|v_3-v_1|,|v_3-v_2|)$
- Appropriate during topology changes
  - Rossignac and Borrel 93
  - Luebke and Erikson 97
- Loose for topology-preserving collapse
Vertex-Plane Distance

- Store set of planes with each vertex
  - Error based on distance from vertex to planes
  - When vertices are merged, merge sets

- Ronfard and Rossignac 96
  - Store plane sets, compute max distance

- Error quadrics - Garland and Heckbert 96
  - Store quadric form, compute sum of squared distances
Point-Surface Distance

- For each original vertex, find closest point on simplified surface
- Compute sum of squared distances
Surface-Surface Distance

- Compute or approximate maximum distance between input and simplified surfaces
  - Tolerance Volumes (Gueziec 96)
  - Simplification Envelopes (Cohen/Varshney 96)
  - Hausdorff Distance (Klein 96)
  - Mapping Distance (Bajaj/Schikore 96, Cohen et al. 97)
Geometric Observations

- Vertex-Vertex and Vertex-Plane distance
  - Fast
  - Low error in practice, but not guaranteed by metric

- Surface-Surface distance
  - Required for guaranteed error bounds

vertex-vertex ≠ surface-surface
Considerations

- Type of input mesh?
- Modifies topology?
- Continuous LOD?
- Speed vs. quality?
View-Dependent Extension

- Simplify dynamically according to viewpoint

Hoppe
View-Dependent Extension

nfaces=213 pixel_tol=0.29
Preserving Appearance

7,809 tris
3,905 tris
1,951 tris
488 tris
975 tris

Caltech & Stanford Graphics Labs and Jonathan Cohen
Remeshing

- Alternative to decimation
- **Placing** polygons to approximate shape vs. greedily **removing** polygons from a complex model
- “Bottom Up” vs. “Top Down”
- Usually better approximation at a low polygon count
- Can place polygons in more “intuitive” places
Anisotropic Remeshing

- Draw lines of curvature, place samples, connect

[Alliez et al., SIGGRAPH 2003]
Variational Approximation

- Grow “close-to-planar” patches and then polygonize

[Cohen-Steiner et al. 2004]