4.1 Polygon Meshes and Implicit Surfaces

Hao Li
http://cs420.hao-li.com
Geometric Representations

- Point based
- Quad mesh
- Triangle mesh
- Implicit surfaces / particles
- Volumetric
- Tetrahedrons
Modeling Complex Shapes

• An equation for a sphere is possible, but how about an equation for a telephone, or a face?

• Complexity is achieved using simple pieces:
  - polygons, parametric surfaces, or implicit surfaces

• Goals
  - Model anything with arbitrary precision (in principle)
  - Easy to build and modify
  - Efficient computations (for rendering, collisions, etc.)
  - Easy to implement (a minor consideration...)

What do we need from shapes in Computer Graphics?

- Local control of shape for modeling
- Ability to model what we need
- Smoothness and continuity
- Ability to evaluate derivatives
- Ability to do collision detection
- Ease of rendering

No single technique solves all problems!
Shape Representations

- Polygon Meshes
- Parametric Surfaces
- Implicit Surfaces
Polygon Meshes

• Any shape can be modeled out of polygons
  – if you use enough of them…

• Polygons with how many sides?
  - Can use triangles, quadrilaterals, pentagons, … n-gons
  - Triangles are most common
  - When > 3 sides are used, ambiguity about what to do
    when polygon nonplanar, or concave, or self-intersecting

• Polygon meshes are built out of
  - vertices (points)
  - edges (line segments between vertices)
  - faces (polygons bounded by edges)
Polygon Models in OpenGL

• for faceted shading

```c
glNormal3fv(n);
glBegin(GL_POLYGONS);
  glVertex3fv(vert1);
  glVertex3fv(vert2);
  glVertex3fv(vert3);
glEnd();
```

• for smooth shading

```c
glBegin(GL_POLYGONS);
  glNormal3fv(normal1);
  glVertex3fv(vert1);
  glNormal3fv(normal2);
  glVertex3fv(vert2);
  glNormal3fv(normal3);
  glVertex3fv(vert3);
glEnd();
```
Triangle defines unique plane
- can easily compute normal
- depends on vertex orientation!
- clockwise order gives

Vertex normals less well defined
- can average face normals
- works for smooth surfaces
- but not at sharp corners
(think of a cube)

Normals

$v_3$
$v_2$
$v_1$

$a = v_2 - v_1$
$b = v_3 - v_1$

$n = \frac{a \times b}{\|a \times b\|}$

$n' = -n$
Where Meshes Come From

• Model manually
  - Write out all polygons
  - Write some code to generate them
  - Interactive editing: move vertices in space

• Acquisition from real objects
  - 3D scanners, vision systems
  - Generate set of points on the surface
  - Need to convert to polygons
Mesh Data Structures

• How to store geometry & connectivity?
• compact storage and file formats
• Efficient algorithms on meshes
  • Time-critical operations
  • All vertices/edges of a face
  • All incident vertices/edges/faces of a vertex
Different Data Structures:

- Different topological data storage
- Most important ones are face and edge-based (since they encode connectivity)
- Design decision ~ memory/speed trade-off
Face Set (STL)

Face:

- 3 vertex positions

<table>
<thead>
<tr>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_{11} y_{11} z_{11}</td>
</tr>
<tr>
<td>x_{21} y_{21} z_{21}</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>x_{F1} y_{F1} z_{F1}</td>
</tr>
</tbody>
</table>

9*4 = 36 B/f (single precision)
72 B/v (Euler Poincaré)

No explicit connectivity
Indexed Face List:

- Vertex: position
- Face: Vertex Indices

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \ y_1 \ z_1$</td>
<td>$i_{11} \ i_{12} \ i_{13}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_v \ y_v \ z_v$</td>
<td>...</td>
</tr>
</tbody>
</table>

$12B/v + 12B/f = 36B/v$

No explicit adjacency info
Face-Based Connectivity

**Vertex:**
- position (12B)
- 1 face (4B)

**Face:**
- 3 vertices (12B)
- 3 face neighbors (24B)

64 B/v

No edges: Special case handling for arbitrary polygons
Edges always have the same topological structure

Efficient handling of polygons with variable valence
(Winged) **Edge-Based Connectivity**

**Vertex:**
- position
- 1 edge

**Edge:**
- 2 vertices
- 2 faces
- 4 edges

**Face:**
- 1 edges

**120 B/v**

Edges have no orientation: special case handling for neighbors
Halfedge-Based Connectivity

Vertex:
- position
- 1 halfedge

Edge:
- 1 vertex
- 1 face
- 1, 2, or 3 halfedges

Face:
- 1 halfedge

96 to 144 B/v

Edges have orientation: No-runtime overhead due to arbitrary faces
Data Structures for Polygon Meshes

• Simplest (but dumb)
  - float triangle[n][3][3]; (each triangle stores 3 (x,y,z) points)
  - redundant: each vertex stored multiple times

• Vertex List, Face List
  - List of vertices, each vertex consists of (x,y,z) geometric (shape) info only
  - List of triangles, each a triple of vertex id’s (or pointers) topological (connectivity, adjacency) info only
    Fine for many purposes, but finding the faces adjacent to a vertex takes $O(F)$ time for a model with $F$ faces. Such queries are important for topological editing.

• Fancier schemes:
  - Store more topological info so adjacency queries can be answered in $O(1)$ time.
  - Winged-edge data structure – edge structures contain all topological info (pointers to adjacent vertices, edges, and faces).
# OBJ file for a 2x2x2 cube

v -1.0  1.0  1.0  - Vertex 1
v -1.0 -1.0  1.0  - Vertex 2
v  1.0 -1.0  1.0  - Vertex 3
v  1.0  1.0  1.0  - ...
v -1.0  1.0 -1.0
v -1.0 -1.0 -1.0
v  1.0 -1.0 -1.0
v  1.0  1.0 -1.0
f  1  2  3  4
f  8  7  6  5
f  4  3  7  8
f  5  1  4  8
f  5  6  2  1
f  2  6  7  3

Syntax:

- `v x y z` - a vertex a (x,y,z)
- `f v1 v2 ... vn` - a face with vertices v1 v2 ... vn
- `#anything` - comment
How Many Polygons to Use?

- 5802 triangles
- 800 triangles
- 300 triangles
- 100 triangles
Why Level of Detail?

- Different models for near and far objects
- Different models for rendering and collision detection
- Compression of data recorded from the real world

- We need automatic algorithms for reducing the polygon count without
  - losing key features
  - getting artifacts in the silhouette
  - popping
Problems with Triangular Meshes?

- Need a lot of polygons to represent smooth shapes
- Need a lot of polygons to represent detailed shapes
- Hard to edit
- Need to move individual vertices
- Intersection test? Inside/outside test?
Shape Representations

- Polygon Meshes
- Parametric Surfaces
- Implicit Surfaces
Parametric Surfaces

$$p(u, v) = [x(u, v), y(u, v), z(u, v)]$$

- e.g. plane, cylinder, bicubic surface, swept surface
Parametric Surfaces

\[ p(u, v) = [x(u, v), y(u, v), z(u, v)] \]

- e.g. plane, cylinder, bicubic surface, swept surface
Parametric Representation

Surface is the range of a function

\[ f : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad S_\Omega = f(\Omega) \]

2D example: A Circle

\[ f : [0, 2\pi] \rightarrow \mathbb{R}^2 \]

\[ f(t) = \begin{pmatrix} r \cos(t) \\ r \sin(t) \end{pmatrix} \]
Parametric Representation

Surface is the range of a function

\[ f : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3, \quad S_{\Omega} = f(\Omega) \]

2D example: Island coast line

\[ f : [0, 2\pi] \to \mathbb{R}^2 \]

\[ f(t) = \left( \begin{array}{c} ? \\ ? \end{array} \right) \]
Surface is the range of a function

$$f : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad \mathcal{S}_\Omega = f(\Omega)$$

2D example: Island coast line

$$f : [0, 2\pi] \rightarrow \mathbb{R}^2$$

$$f(t) = \left( \begin{array}{c} r \cos(t) \\ r \sin(t) \end{array} \right)$$
Polygonal meshes are a good compromise

- Piecewise linear approximation $\rightarrow$ error is $O(h^2)$
Polygonal meshes are a good compromise

- Piecewise linear approximation $\rightarrow$ error is $O(h^2)$
- Error inversely proportional to #faces
Polygonal meshes are a good compromise

- Piecewise linear approximation $\rightarrow$ error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
- Piecewise smooth surfaces
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling
- Efficient GPU-based rendering/processing
Parametric Surfaces

• Why better than polygon meshes?
  - Much more compact
  - More convenient to control --- just edit control points
  - Easy to construct from control points

• What are the problems?
  - Work well for smooth surfaces
  - Must still split surfaces into discrete number of patches
  - Rendering times are higher than for polygons
  - Intersection test? Inside/outside test?
Shape Representations

- Polygon Meshes
- Parametric Surfaces
- Implicit Surfaces
Two Ways to Define a Circle

**Parametric**

\[ x = f(u) = r \cos(u) \]
\[ y = g(u) = r \sin(u) \]

**Implicit**

\[ F(x, y) = x^2 + y^2 - r^2 \]
Implicit Surfaces

• well defined inside/outside
• polygons and parametric surfaces do not have this information
• Computing is hard:
  - implicit functions for a cube? telephone?

• Implicit surface: \( F(x, y, z) = 0 \)
  - e.g. plane, sphere, cylinder, quadric, torus, blobby models
  sphere with radius \( r \) : \( F(x, y, z) = x^2 + y^2 + z^2 - r^2 = 0 \)
  - terrible for iterating over the surface
  - great for intersections, inside/outside test
Quadric Surfaces

\[ F(x, y, z) = ax^2 + by^2 + cz^2 + 2fyz + 2hxy + 2px + 2qy + 2rz + d = 0 \]
What Implicit Functions are Good For

Ray - Surface Intersection Test

Inside/Outside Test

\[ F(x + kv) = 0 \]
Surfaces from Implicit Functions

- Constant Value Surfaces are called (depending on whom you ask):
  - constant value surfaces
  - level sets
  - isosurfaces

- Nice Feature: you can add them! (and other tricks)
  - this merges the shapes
  - When you use this with spherical exponential potentials, it’s called *Blobs*, *Metaballs*, or *Soft Objects*. Great for modeling animals.
Blobby Models

by Brian Wyvill, http://www.cpsc.ucalgary.ca/~blob/
How to draw implicit surfaces?

• It’s easy to ray trace implicit surfaces
  - because of that easy intersection test

• Volume Rendering can display them

• Convert to polygons: the Marching Cubes algorithm
  - Divide space into cubes
  - Evaluate implicit function at each cube vertex
  - Do root finding or linear interpolation along each edge
  - Polygonize on a cube-by-cube basis
Constructive Solid Geometry (CSG)

- Generate complex shapes with basic building blocks
- Machine an object - saw parts off, drill holes, glue pieces together
Constructive Solid Geometry (CSG)

union
the merger of two objects into one

difference
the subtraction of one object from another

intersection
the portion common to both objects
Constructive Solid Geometry (CSG)

- Generate complex shapes with basic building blocks
- Machine an object - saw parts off, drill holes, glue pieces together
- This is sensible for objects that are actually made that way (human-made, particularly machined objects)
A CSG Train
Negative Objects

- Use point-by-point boolean functions
  - remove a volume by using a negative object
  - e.g. drill a hole by subtracting a cylinder

\[
\text{Inside}(\text{BLOCK-CYL}) = \text{Inside}(\text{BLOCK}) \text{ And } \text{Not}(\text{Inside}(\text{CYL}))
\]
Set Operations

- **UNION:**
  - Inside(A) || Inside(B)
  - Join A and B

- **INTERSECTION:**
  - Inside(A) && Inside(B)
  - Chop off any part of A that sticks out of B

- **SUBTRACTION:**
  - Inside(A) && (! Inside(B))
  - Use B to Cut A

Examples:
- Use cylinders to drill holes
- Use rectangular blocks to cut slots
- Use half-spaces to cut planar faces
- Use surfaces swept from curves as jigsaws, etc.
Implicit Functions for Booleans

• Recall the implicit function for a solid: $F(x,y,z)<0$

• Boolean operations are replaced by arithmetic
  - $\text{MAX}$ replaces $\text{And}$ (intersection)
  - $\text{MIN}$ replaces $\text{OR}$ (union)
  - $\text{MINUS}$ replaces $\text{NOT}$ (unary subtraction)

• Thus
  - $F(\text{Intersect}(A,B)) = \text{MAX}(F(A),F(B))$
  - $F(\text{Union}(A,B)) = \text{MIN}(F(A),F(B))$
  - $F(\text{Subtract}(A,B)) = \text{MAX}(F(A), -F(B))$
CSG Trees

• Set operations yield tree-based representation

Implicit Surfaces

- Good for smoothly blending multiple components
- Clearly defined solid along with its boundary
- Intersection test and Inside/outside test are easy
- Need to polygonize to render --- expensive
- Interactive control is not easy
- Fitting to real world data is not easy
- Always smooth
Summary

- Polygon Meshes
- Parametric Surfaces
- Implicit Surfaces
- Constructive Solid Geometry
http://cs420.hao-li.com

Thanks!