Polygon Meshes and Implicit Surfaces

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!

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...)

Shape Representations

Polygon Models in OpenGL

- for faceted shading
  ```
  glBegin(GL_TRIANGLES);
  glVertex3f(vert1);  glVertex3f(vert2);  glVertex3f(vert3);
  glEnd();
  ```
- for smooth shading
  ```
  glBegin(GL_TRIANGLES);
  glVertex3f(vert1);
  glVertex3f(vert2);
  glVertex3f(vert3);
  glEnd();
  ```

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

Normals

Triangle defines unique plane
- can easily compute normal
\[ \mathbf{n} = \mathbf{b} \times \mathbf{c} \]
- depends on vertex orientation
- clockwise order gives
\[ u' = -u \]

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

Where Meshes Come From

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

- Acquisition from real objects
  - Laser scanners, vision systems
  - Generate set of points on the surface
  - Need to convert to polygons

Data Structures for Polygon Meshes

- Simplest (but dumb)
  - float triangle[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)

A File Format for Polygon Models: OBJ

# OBJ file for a 2x2x2 cube
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 5 6 7 8
f 9 10 11 12
f 13 14 15 16

Syntax:
- a vertex at (x,y,z)
v x y z
- a face with vertices v1, v2, ... vn
f v1 v2 ... vn
# anything - comment

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?

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

The Utah teapot

Shape Representations

Polygon Meshes
Parametric Surfaces
Implicit 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?
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

\[ F(x,y,z) = ax^2 + by^2 + cz^2 + 2fyz + 2gxz + 2hxy + 2pz + 2qy + rz + d = 0 \]

What Implicit Functions are Good For

Ray - Surface Intersection Test
Inside/Outside Test

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 Dicer (W3D), http://www.cse.ucsd.edu/~dicer/
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)

- This is sensible for objects that are actually made that way (human-made, particularly machined objects)

Constructive Solid Geometry (CSG)

- Generate complex shapes with basic building blocks
- Machine an object - saw parts off, drill holes, glue pieces together

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

Inside(BLOCK-CYL) = Inside(BLOCK) And Not(Inside(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:
  - MAX replaces AND (intersection)
  - MIN replaces OR (union)
  - MINUS replaces 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

- Polygonal Meshes
- Parametric Surfaces
- Implicit Surfaces
- Constructive Solid Geometry