Announcements

• Graded:
  – Programming Assignment 1 – Ian or Michael
  – Grades in R6 in your turnin directory
  – Written Assignment – Michael
  – Derivation for Assignment 2 – Ian
• Programming Assignment 2 due on Thursday – questions?
• Written Assignment 2 out on Thursday

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 one technique solves all problems

Polygon Meshes
Implicit Surfaces
Constructive Solid Geometry

Watt: Chapter 2

Two Ways to Define a Circle

Parametric

Implicit

Curve Representations

• Explicit: \( y = f(x) \)
  \( y = mx + b \)
  \( y = x^2 \)

• Parametric: \((x,y) = (f(u),g(u))\)
  \((x,y) = (\cos(u), \sin(u))\)

• Implicit: \( f(x,y) = 0 \)
  \( x^2 + y^2 - r^2 = 0 \)

Surface Representations

• Parametric surface — \( x(u,v), y(u,v), z(u,v) \)
  – e.g. plane, sphere, cylinder, torus, bicubic surface, swept surface
  – parametric functions let you iterate over the surface by incrementing \( u \) and \( v \)
  – great for making polygon meshes, etc.
  – terrible for intersections: ray/surface, point-inside-boundary, etc.
• Implicit surface: \( F(x,y,z) = 0 \)
  – e.g. plane, sphere, cylinder, quadric, blobby models
  – terrible for iterating over the surface
  – great for intersections, morphing
• Subdivision surfaces
  – defined by a control mesh and a recursive subdivision procedure
  – good for interactive design
Modeling Complex Shapes

- We want to build models of very complicated objects.
- An equation for a sphere is possible, but how about an equation for a telephone, or a face, or a cloud?
- 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...)

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)

Frontfacing / Backfacing

- A polygon has two sides, of course.
- Customary in CG to use the right hand rule to pick one side to call the front face.
- Counterclockwise = front, clockwise = back.
- Important for:
  - Lighting
  - Backface culling
  - For the triangle ABC below, the front face is up.

Normals and Plane Equations

- Need normals for shading, plane eqns for intersection tests.
- A normal to a plane is a vector that is perpendicular to that plane (two possible choices).
- A plane is specified by a point P and a normal vector N.
- N*(X-P) = 0 if and only if X lies in the plane; this is an implicit equation for the plane.
- Expand this out: N*(a*x + b*y + c*z + d) = 0.
- 3 vertices define a plane, its normal is: N = (B-A) x (C-A).
- Unit normal: N = N/||N||.

Polygon Models in OpenGL

- For faceted shading:
  < calculate face normal n using cross product rule >
  glBegin(GL_POLYGON); glVertex3fv(vert1); glVertex3fv(vert2); glVertex3fv(vert3); glEnd();
- For smooth shading:
  glBegin(GL_POLYGON); glNormal3fv(normal1); glVertex3fv(vert1); glNormal3fv(normal2); glVertex3fv(vert2); glNormal3fv(normal3); glVertex3fv(vert3); glEnd();

Data Structures for Polygon Meshes

- Simplest (but dumb):
  - 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.
  - Fast for many purposes, but locating the faces adjacent to a vertex (take O(F) time for a model with F faces). Such queries are important for topological editing.
- Fancier schemes:
  - Winged-edge data structure: Edge structures contain all topological info (pointers to adjacent vertices, edges, and faces).
  - 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

Syntax:
- `v x y z` - a vertex at (x,y,z)
- `f` - a face with vertices v_1, v_2, ..., v_n
- `#` - anything - comment

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

How Many Polygons to Use?

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, etc.

Surface Representations
- Parametric surface — x(u,v), y(u,v), z(u,v)
  - e.g. plane, cylinder, bicubic surface, swept surface
  - parametric functions let you iterate over the surface by incrementing u and v in nested loops
  - great for making polygon meshes, etc
  - terrible for intersections: ray/surface, point-inside-boundary, etc.
- Implicit surface: F(x,y,z) = 0
  - e.g. plane, sphere, cylinder, quadric, torus, blobby models
  - terrible for iterating over the surface
  - great for intersections, morphing

Sets of Points, Surfaces and Solids
- Implicit surface: set of all points that satisfy F(x,y,z)=0
  - Points that satisfy F(x,y,z)=0 define a solid (or solids) bounded by the surface
  - The solid is directly defined (unlike definitions using parametric surfaces)
  - Example
    - An infinitely long (solid) cylinder with radius r:
    - To limit cylinder to length L, abs(z) < L/2 and keep the function implicit use max:
    - Implicit functions for a cube? Any convex polyhedron?

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
• Implicit function is the sum of Gaussians centered at several points in space, minus a threshold
• varying the standard deviations of the Gaussians makes each blob bigger
• varying the threshold makes blobs merge or separate

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

Isosurfaces of Simulated Tornado

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
Brian Wyvill & students, Univ. of Calgary
Negative Objects

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

\[ \text{Inside(BLOCK-CYL)} = \text{Inside(BLOCK) And \neg (\text{Inside(CYL)})} \]  

Set Operations

- UNION: \[ \text{Inside(A)} \lor \text{Inside(B)} \]
  - Join A and B
- INTERSECTION: \[ \text{Inside(A)} \land \text{Inside(B)} \]
  - Chop off any part of A that sticks out of B.
- SUBTRACTION: \[ \text{Inside(A)} \land (\neg \text{Inside(B)}) \]
  - 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:
  - MINUS replaces \( \neg \) (unary subtraction)
  - MAX replaces AND (intersection)
  - MIN replaces OR (union)

Thus:
- \( F(\text{Subtract}(A,B)) = \max(F(A), -F(B)) \)
- \( F(\text{Intersect}(A,B)) = \max(F(A), F(B)) \)
- \( F(\text{Union}(A,B)) = \min(F(A), F(B)) \)

You can try this at home

- Drawing boolean objects - combine parametric and implicit functions
- The boolean object has surfaces from all its constituent objects
- Draw using polygonal meshes, test before drawing using implicit function
- For a hole drilled in a block, the surface of the hole is given by the cylinder used to drill it
- Draw points on the block if they are outside the cylinder
- Draw points on the cylinder if they are inside the block
- Implementing union:
  - draw both objects, use hidden-surface algorithms to take care of visibility
- Implementing intersection:
  - draw points only if they are inside both objects
- Implementing subtraction:
  - points on the positive object’s surface are visible outside the negative object
  - points on the negative object’s surface are visible inside the positive object
- Draw using parametric functions, trim using implicit functions
- And that’s where the tricky part comes in.

3-D Object Representation

- Individual elements are voxels (volume elements)
- Compression is almost mandatory
- Use octrees (3-D version of quadtree)
  - simply subdivide a cube into 8 sub-cubes forming a tree
  - stop subdividing when the whole cube is entirely full or empty, or the minimum resolution is reached
  - at minimum resolution fill the block if majority is full
  - combine empty cubes if they all have the same state
- Partially full cubes are nodes, full or empty cubes are leaves
- Data space requirement is proportional to the surface area of the object (except a few worst cases)

Announcements

- Graded:
  - Programming Assignment 1 – Ian or Michael
  - Written Assignment – Michael
  - Derivation for Assignment 2 – Ian
- Programming Assignment 2 due on Thursday – questions?
- Written Assignment 2 out on Thursday