Lecture 22 – 3D Modeling & Polygons

Michael Gleicher
November, 2006
Used as notes, not projected as lecture

Modeling 3D Shapes

• Modeling = process of describing an object
  – Representation
• Can model shape, physical properties, behavior, …

• Many uses of (geometric) models
  – Graphics – make a picture
  – CAD – represent for manufacture

Types of Shape Models in 3D

• Points
• Curves
• Surfaces and Solids
• Volumes

Surface vs. Volume

• Cube
  – Volume = space inside 0 <= x,y,z < 1
  – Surface = 6 squares (0,0,0)(0,0,1)(0,1,1)…
• Surface can be a boundary
  – But might not be

• Graphics (often) only need surfaces

When might we care about Volumes?

• Engineering / Manufacturing / Design
  – Can’t be non-physical
• Some kinds of data has “insides”
  – Medical data (scanned)
• Some operations make sense
  – Constructive solid geometry
  – Cut / Join / Subtract / Union
  – Makes less sense on surfaces

How to do volumes?

• Hard: need to insure that you always have a volume!
• Operations on primitives
  – Make solid pieces (spheres, cylinders, polyhedra, …)
  – Combine with sensible operators (union, intersection, difference)
• Construction Solid Geometry
• Boundary Representations
  – Store the surface
  – Represent what’s inside
  – Be careful that there always is an inside – no holes!
• Implicit Representations
  – F(x) < 0 – for some fancy F
  – Distance fields, union of blobs, …
  – Tend to be special purpose
• Sampled Volumes (like medical data)
Surface Basics
- Locally flat
- At any point
  - Normal
  - Tangent Plane
  - Tangent vectors in plane

Surfaces
- Generally what we use in graphics
  - Hard enough!
- Similar issues to curves, but worse
- Named vs. Free-Form
- Build out of little pieces
- Linear pieces (polygons) – analogy to lines

Basic Strategy
- Break complicated surfaces into pieces
- Need to choose good pieces
- Need to make sure that the pieces connect
- Connections are more complicated

Polygons
- Or triangles
- Need to have a front/back
- Outward facing normal
- Be consistent in orientation (e.g. CCW)

Polygon Soup
- Random Assortment
- Unstructured
  - At least get ordering right
- Tells little about how polygons connect
- Lots of redundancy

Cube Soup
struct Triangle Cube[12] =
{{(1,1,1),(1,0,0),(1,1,0)},
{(1,1,1),(1,0,1),(1,0,0)},
{(0,1,1),(1,1,1),(0,1,0)},
{(1,1,1),(1,1,0),(0,1,0)},
...
};
Polygon Soup

• Advantages
  – Easy
• Problems
  – Redundancy
  – No global info
  – No open/closed info
  – Hard to edit
  – Hard to prevent degeneracies
  – No non-local information
  – Is it closed?
  – Is it connected?
  – Is this an edge or internal?

Cracks / Cracking

• Gaps in the surface
• Prevents from being solid
• Can be ugly
• Airtight / Watertight
  – No cracks
• Beware edge/vertex
  – Numerical errors cause cracks

Mesh

• Share vertices
  – Indirection to vertex table
  – Prevents cracking
  – More efficient (lots of info at vertex)
• Store Polygons as vertex lists
• Store Edges – Faces are lists of edges
  – Every edge borders 2 faces
• Simplicial Complex
  – Mathematically deep term
  – Fancy way to say “nice mesh” – all faces meet at an edge, …

Vertex Indirection

• List of vertices
• Everything is an index into this table
• Good points:
  – Sharing prevents some cracking
  – Transform/Light each vertex once
  – Data reduction

More complex Mesh Structures

• Store Edges
  – Can be handy to have
• Each edge only 2 faces – one CW one CCW (pass through edge in opposite ways)
  – Store “next” edge for each direction
  – Winged Edge Data Structure

Getting Meshes to Hardware Fast

• Minimize number of vertices sent down pipe
  – Old days – definitely bottleneck
  – Now – maybe not, since lots of per-pixel computation
• Vertex Buffers
  – Send small number of vertices
  – Index into this small array (since memory << model size)
  – Group into small sets (like 8 or 16) of vertices, draw all triangles between them
• Vertex Cache
  – Automatically buffer, use LIFO
Vertex Arrays

- Hardware caches vertices (after transform)
- Give vertex list and connectivity
- Do in an order to get cache performance
  - Groups of n vertices
- Hardware specific trick

- Best way to draw triangles in opengl
- Send blocks of data at once (avoid function call overhead)
  - Can be high since function call means call to low-level driver
- Possibly: store array in fast memory specific for graphics
  - On graphics card or in driver address space
- Issues with data formats

Regular Meshes

- Reduce number of vertices needed
- Reduce amount of connectivity info needed (which can be sizable)

- Often have meshes with uniform patterns
- Grids, fans, strips
- Connectivity is implicit
- Very efficient
- Processing is easy
- Avoid redundant transforms