Simple Mesh Format (SMF)

- Michael Garland
  http://graphics.cs.uiuc.edu/~garland/
- Triangle data
  - Vertex indices begin at 1

3D Clipping

- Cohen-Sutherland and Cyrus-Beck can be trivially extended to 3D
- We will cover:
  - Cohen-Sutherland for 3D, (parallel projection)
  - Cohen-Sutherland for 3D, (perspective projection)

Recall: Cohen-Sutherland

- Line is completely visible iff both code values of endpoints are 0, i.e. $C_0 \lor C_1 = 0$
- If line segments are completely outside the window, then $C_0 \land C_1 \neq 0$

Cohen-Sutherland for 3D, Parallel Projection

- Use 6 bits
- Trivially accept if all end-codes are 0
- Trivially reject if bit-by-bit AND of end-codes is not 0
- Up to 6 intersections may have to be computed

Cohen-Sutherland for 3D computing intersection points.

- Use parametric representation of the line to compute intersections
- So for $y=1$ replace $y$ with 1 and solve for $t$
- If $1 \geq t \geq 0$ use it to find $x$ and $z$
- Test if $x$ and $z$ are in valid range
- Repeat for planes $y=-1$, $x=1$, $x=-1$, $z=-1$, $z=0$
Cohen-Sutherland for 3D, Perspective Projection

- Use 6 bits identical to parallel view volume clipping
- Conditions on the codes are different
- Trivially accept/reject lines using same roles
- Intersection points computed differently

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>point ABOVE the view volume</td>
<td>( y &gt; -z )</td>
</tr>
<tr>
<td>2</td>
<td>point BELOW the view volume</td>
<td>( y &lt; z )</td>
</tr>
<tr>
<td>3</td>
<td>point RIGHT OF the view volume</td>
<td>( x &gt; -z )</td>
</tr>
<tr>
<td>4</td>
<td>point LEFT OF the view volume</td>
<td>( x &lt; z )</td>
</tr>
<tr>
<td>5</td>
<td>point BEHIND the view volume</td>
<td>( z &lt; -1 )</td>
</tr>
<tr>
<td>6</td>
<td>point IN FRONT the view volume</td>
<td>( z &gt; z_{\text{min}} )</td>
</tr>
</tbody>
</table>

“3D Clipping” for HWs 4 & 5

- Only do trivial reject test
- For HW4 just do X and Y tests
- ‘AND’ all vertex bit codes for a polygon
- If result \(!= 0\), then reject polygon
  - i.e. remove from projection pipeline

More Efficient Alternative?

- Use 3D Cohen-Sutherland to do trivial reject
- Project remaining polygons onto view plane
- Clip polygons in 2D
- Remember that user-defined window is redefined for canonical view volumes!

Overview

- 3D model representations
- Mesh formats
- Bicubic surfaces
- Bezier surfaces
- Normals to surfaces
- Direct surface rendering
3D Modeling

- 3D Representations
  - Wireframe models
  - Surface Models
  - Solid Models
  - Meshes and Polygon soups
  - Voxel/Volume models
  - Decomposition-based
    - Octrees, voxels
- Modeling in 3D
  - Constructive Solid Geometry (CSG), B-reps and feature-based

Representing 3D Objects

- Exact
  - Wireframe
  - Parametric Surface
  - Solid Model
    - CSG
    - BRep
    - Implicit Solid Modeling
- Approximate
  - Facet / Mesh
    - Just surfaces
  - Voxel
    - Volume info

Negatives when Representing 3D Objects

- Exact
  - Complex data structures
  - Expensive algorithms
  - Wide variety of formats, each with subtle nuances
  - Hard to acquire data
  - Translation required for rendering
- Approximate
  - Lossy
  - Data structure sizes can get HUGE, if you want good fidelity
  - Easy to break (i.e. cracks can appear)
  - Not good for certain applications
  - Lots of interpolation and guess work

Positives when Representing 3D Objects

- Exact
  - Precision
    - Simulation, modeling, etc.
  - Lots of modeling environments
  - Physical properties
  - High-level control
  - Many applications (tool path generation, motion, etc.)
  - Compact
- Approximate
  - Easy to implement
  - Easy to acquire
    - 3D scanner, CT
  - Easy to render
    - Direct mapping to the graphics pipeline
  - Lots of algorithms

Exact Representations

- Wireframe
- Parametric Surface
- Solid Model
  - operations
    - CSG, BRep, implicit geometry
Wireframes

• Basic idea:
  – Represent the model as the set of all of its edges

• Example:
  A simple cube
  – 12 lines
  – 8 vertices

• How about the faces?

Issues with Wireframes

• Visually ambiguous
  • No surfaces!
    – What’s inside? What’s outside?
    – Hidden line removal?

• What does validity entail?
  – Don’t we just have a bunch of wires?
  – Do they need to add up to something?

• How to model wireframe shapes?
  – Wire by wire? Not very easy!

Surface Models

• Basic idea:
  – Represent a model as a set of faces/patches

• Limitations:
  – Topological integrity; how do faces “line up”; which way is ‘inside’ / ‘outside’?

• Used in many CAD applications
  – Why? They are fine for drafting and rendering, not as good for creating true physical models

3D Mesh File Formats

Some common formats

• STL
• SMF
• OpenInventor
• VRML
• X3D

Minimal

• Vertex + Face

• No colors, normals, or texture

• Primarily used to demonstrate geometry algorithms

Full-Featured

• Colors / Transparency
• Vertex-Face Normals (optional, can be computed)
• Scene Graph
• Lights
• Textures
• Views and Navigation
Simple Mesh Format (SMF)

- Michael Garland
  - http://graphics.cs.uiuc.edu/~garland/
- Triangle data
- Vertex indices begin at 1
- #SMF 1.0
- #directions 5
- #faces 6
  - v 2.0 0.0 2.0
  - v 2.0 0.0 -2.0
  - v -2.0 0.0 -2.0
  - v -2.0 0.0 2.0
  - v 0.0 5.0 0.0
  - e 1 3 5
  - e 2 4 5
  - e 3 5 2
  - e 2 5 1
  - e 1 5 4
  - e 4 5 3

Stereolithography (STL)

- Triangle data + Face Normal
- The de-facto standard for rapid prototyping
- solid
- ...
- front normal 0.00 0.00 1.00
- begin loop
- vertex 2.00 2.00 0.00
- vertex -1.00 1.00 0.00
- vertex 0.00 -1.00 0.00
- endloop
- endfacet
- ...
- endobj

Open Inventor

- Developed by SGI
- Predecessor to VRML
  - Scene Graph

Virtual Reality Modeling Language (VRML)

- SGML Based
- #VRML V2.0 utf8
- #A Cylinder
- Shape
  - appearance Appearance { material Material ()
    geometry Cylinder {
      height 2.0
      radius 1.5
    }
  }

X3D

- Open standards file format and run-time architecture to represent and communicate 3D scenes and objects using XML
- Supports
  - 2D/3D graphics, programmable shaders
  - 2D/3D compositing, CAD data, Animation
  - Spatialized audio and video, User interaction
  - Navigation, Scripting, Networking, Simulation
- See www.web3d.org for more info

Issues with 3D “mesh” formats

- Easy to acquire
- Easy to render
- Harder to model with
- Error prone
  - split faces, holes, gaps, etc
BRep Data Structures

- Winged-Edge Data Structure (Weiler)
- Vertex
  - n edges
- Edge
  - 2 vertices
  - 2 faces
- Face
  - m edges

Biparametric Surfaces

- Biparametric surfaces
  - A generalization of parametric curves
  - 2 parameters: s, t (or u, v)
  - Two parametric functions

Bicubic Surfaces

- Recall the 2D curve: \( Q(s) = G \cdot M \cdot S \)
  - G: Geometry Matrix
  - M: Basis Matrix
  - S: Polynomial Terms \( [s^3 \ s^2 \ s \ 1] \)
- For 3D, we allow the points in G to vary in 3D along t as well:
  \[
  Q(s, t) = \begin{bmatrix}
  G_1(t) & G_2(t) & G_3(t) & G_4(t)
  \end{bmatrix} \cdot M \cdot S
  \]

Observations About Bicubic Surfaces

- For a fixed \( t_1 \), \( Q(s, t_1) \) is a curve
- Gradually incrementing \( t_1 \) to \( t_2 \), we get a new curve
- The combination of these curves is a surface
- \( G_i(t) \) are 3D curves
Bicubic Surfaces

- Each \( G_i(t) \) is \( G_i(t) = G_i \cdot M \cdot T \), where
  \[
  G_i = \begin{bmatrix}
  g_{i1} & g_{i2} & g_{i3} & g_{i4}
  \end{bmatrix}
  \]
- Transposing \( G_i(t) \), we get
  \[
  G_i(t) = T^T \cdot M^T \cdot G_i^T
  = T^T \cdot M^T \cdot g_{i1} \begin{bmatrix}
  g_{i2} & g_{i3} & g_{i4}
  \end{bmatrix}
  \]

Bicubic Surfaces

- Substituting \( G_i(t) \) into \( Q(s) = G \cdot M \cdot S \), we get \( Q(s, t) \)
- The \( g_{it} \) etc. are the control points for the Bicubic surface patch:
  \[
  Q(s, t) = T^T \cdot M^T \cdot \begin{bmatrix}
  g_{11} & g_{21} & g_{31} & g_{41} \\
  g_{12} & g_{22} & g_{32} & g_{42} \\
  g_{13} & g_{23} & g_{33} & g_{43} \\
  g_{14} & g_{24} & g_{34} & g_{44}
  \end{bmatrix} \cdot M \cdot S
  \]

Bézier Surfaces

- Writing out \( Q(s, t) = T^T \cdot M^T \cdot G \cdot M \cdot S \) gives
  \[
  x(s, t) = T^T \cdot M^T \cdot G_X \cdot M \cdot S
  \\
y(s, t) = T^T \cdot M^T \cdot G_Y \cdot M \cdot S
  \\
z(s, t) = T^T \cdot M^T \cdot G_Z \cdot M \cdot S
  \]

Bicubic Bezier Patches

Using same data array \( P = [p_{ij}] \) as with interpolating form

\[
\tilde{p}(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} b_i(u)b_j(v)\tilde{p}_{ij} = u^TPM^{\top}v
\]

Patch lies in convex hull

Bicubic Bézier Patches

- Expanding the summation
  \[
  \tilde{p}(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} b_i(u)b_j(v)\tilde{p}_{ij} = b_{00}(u)b_{00}(v)\tilde{p}_{00} + b_{01}(u)b_{01}(v)\tilde{p}_{01} + b_{02}(u)b_{02}(v)\tilde{p}_{02} + b_{03}(u)b_{03}(v)\tilde{p}_{03} + b_{10}(u)b_{10}(v)\tilde{p}_{10} + etc.
  \]
Cubic Bezier Blending Functions

\[ b(u) = \begin{bmatrix} (1-u)^3 \\ 3u(1-u)^2 \\ 3u^2(1-u) \\ u^3 \end{bmatrix} \]

Note that all zeros are at 0 and 1 which forces the functions to be smooth over (0,1)

Plotting Isolines

Faceting

Composite Bézier Surfaces

- \( C^0 \) and \( G^0 \) continuity can be achieved between two patches by setting the 4 boundary control points to be equal
- \( G^1 \) continuity achieved when cross-wise CPs are co-linear

Bézier Surfaces: Example

- Utah Teapot modeled by 32 Bézier Patches with \( G^1 \) continuity
Beziers Surface: Example

- Increased facet resolution
- Rendered

B-spline Surfaces

\[ x(s,t) = T^T \cdot M_B^s \cdot G_M^t \cdot M_B^t \cdot S \]
\[ y(s,t) = T^T \cdot M_B^s \cdot G_M^t \cdot M_B^t \cdot S \]
\[ z(s,t) = T^T \cdot M_B^s \cdot G_M^t \cdot M_B^t \cdot S \]

- Representation for B-spline patches
- \( C^2 \) continuity across boundaries is automatic with B-splines

Normals to Surfaces

- Normals used for
  - Shading
  - Interference detection in robotics
  - Calculating offsets for numerically controlled machining

Computing the Normals to Surfaces

- For a bicubic surface, first, compute the \( s \) tangent vector:
  \[ \frac{\partial}{\partial s} Q(s,t) \]
  \[ = \frac{\partial}{\partial t} (T^T \cdot M^t \cdot G \cdot M \cdot S) \]
  \[ = T^T \cdot M^t \cdot G \cdot M \cdot \frac{\partial}{\partial s} (S) \]
  \[ = T^T \cdot M^t \cdot G \cdot M \cdot \left[ \begin{array}{ccc} 3t^2 & 2t & 1 \\ 0 & 1 & 0 \end{array} \right] \]

- Next, compute the \( t \) tangent vector:
  \[ \frac{\partial}{\partial t} Q(s,t) \]
  \[ = \frac{\partial}{\partial t} (T^T \cdot M^t \cdot G \cdot M \cdot S) \]
  \[ = T^T \cdot M^t \cdot G \cdot M \cdot S \]
  \[ = \left[ \begin{array}{ccc} 3t^2 & 2t & 1 \\ 0 & 1 & 0 \end{array} \right] \cdot T^T \cdot M^t \cdot G \cdot M \cdot S \]

- Since \( s \) and \( t \) are tangent to the surface, their cross product is the normal vector to the surface:
  \[ \frac{\partial}{\partial s} Q(s,t) \times \frac{\partial}{\partial t} Q(s,t) = \left[ \begin{array}{ccc} y_t z_s - y_s z_t & z_t x_s - z_s x_t & x_t y_s - x_s y_t \end{array} \right] \]

- \( x_t \) - x component of \( s \) tangent
- \( y_t \) - y component of \( s \) tangent
- \( z_t \) - z component of \( s \) tangent
Surface of Revolution

• Rotate planar curve (directrix) around an axis of revolution (z axis)
  – Cross-section is a circle
• Biparametric surface
  – u of curve
  – θ of angle of rotation
• Examples: cylinder, cone, sphere, torus

Drawing Parametric Surfaces

• Usually done “patch by patch”
• Two choices
  – Draw/render directly from the parametric description
  – Approximate the surface with a polygon mesh, then draw/render the mesh

Direct Rendering

• Use a scan-line algorithm
  – Evaluate pixel by pixel
  – Problem: How to go from (x,y) “screen space” to point on the 3D patch
    • Easy for a planar polygon where we know max/min y, equations for edges, screen depth
    • Not as easy for parametric surfaces

Issues for Direct Rendering

• Max/Min y coords may not lie on boundaries
• Silhouette edges result from patch bulges
  – Need to track both silhouettes and boundaries
    • What if they intersect?
    • Note: patch edges need not be monotonic in x or y
• Idea: Scan convert patch plane-by-plane, using scan planes instead of scan lines

Direct Scan Conversion of Patches

• Basic idea
  – Find intersection of patch with XZ plane
    • Producing a planar curve
    • De Boor, D’Casteljeau
  – Draw the curve
    • Note: if doing rendering, one can compute pixel-by-pixel color values this way
  – Patch: x=X(u,v), y=Y(u,v), z=Z(u,v)
Patch to Polygon Conversion

Two methods:
• **Object Space Conversion**
  – Techniques
    • Uniform subdivision
    • Non-uniform subdivision
  – Resolution: depends on object space
• **Image Space Conversion**
  – Resolution: depends on pixels and screen

Object Space Conversion: Uniform Subdivision

Basic Procedure
• Cut parameter space into equal parts
• Find new points on the surface
• Recurse/Repeat “until done”
• Split squares into triangles
• Render

Object Space Conversion: Non-Uniform Subdivision

• Basic idea
  – More facets in areas of high curvature
  – Use change in normals to surface to assess curvature
  – More derivatives
  – Break patch into sub-patches based on curvature changes

Image Space Conversion

• Idea: control subdivision based on screen criteria
  – Minimum pixel area
    • Stop when patch is basically one pixel
  – Screen flatness
    • Stop when patch converges to a polygon
    • Stop when edge is straight or size of pixel

How do I know if I’ve found a silhouette edge?

• If the viewing ray is tangent to the surface at the point it hits the surface!
  \[ \mathbf{N} \cdot \mathbf{L} = 0 \]
  – Where \( \mathbf{N} \) is the normal at the point where \( \mathbf{L} \), the line of sight, hits the surface

Silhouette Determination

Xu, et al., U. of Minnesota
Kawasaki, et al.