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), Brep 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

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
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

Exact Representations

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

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

Stereolithography (STL)
- Triangle data + Face Normal
- The de-facto standard for rapid prototyping

Open Inventor
- Developed by SGI
- Predecessor to VRML
  - Scene Graph

Virtual Reality Modeling Language (VRML)
- SGML Based
- Scene-Graph
- Full Featured
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

BRep Data Structure

- Vertex structure
  - X, Y, Z point
  - Pointers to n coincident edges
- Face structure
  - Pointers to m edges
- Edge structure
  - 2 pointers to end-point vertices
  - 2 pointers to adjacent faces
  - Pointer to next edge
  - Pointer to previous edge

Biparametric Surfaces

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

Biparametric Patch

- (u, v) pair maps to a 3D point on patch
  \[ F(u,v) = (x,y,z) = (x(u,v), y(u,v), z(u,v)) \]
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 increamenting \( t_1 \) to \( t_2 \), we get a new curve
- The combination of these curves is a surface
- \( G_i(t) \) are 3D curves

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
\[
G_i(t) = T^T \cdot M^T \cdot G_i^T
\]
\[
= T^T \cdot M^T \cdot \begin{bmatrix} g_{i1} & g_{i2} & g_{i3} & g_{i4} \end{bmatrix}^T
\]

Substituting \( G_i(t) \) into \( Q(s) = G \cdot M \cdot S \), we get \( Q(s, t) \)
- The \( g_{i1}, \) etc. are the control points for the Bicubic surface patch:
  \[
  Q(s, t) = T^T \cdot M^T \cdot \begin{bmatrix} g_{i1} & g_{i2} & g_{i3} & g_{i4} & g_{i5} & g_{i6} & g_{i7} & g_{i8} \end{bmatrix} \cdot M \cdot S
  \]

Writing out \( Q(s, t) \) gives
\[
\begin{align*}
x(s, t) &= T^T \cdot M^T \cdot G_x \cdot M \cdot S \\
y(s, t) &= T^T \cdot M^T \cdot G_y \cdot M \cdot S \\
z(s, t) &= T^T \cdot M^T \cdot G_z \cdot M \cdot S
\end{align*}
\]

Bicubic Bézier Patch

- Bézier Surfaces (similar definition)
  \[
  x(s, t) = T^T \cdot M_B^T \cdot G_{B_i} \cdot M_B \cdot S \\
y(s, t) = T^T \cdot M_B^T \cdot G_{B_i} \cdot M_B \cdot S \\
z(s, t) = T^T \cdot M_B^T \cdot G_{B_i} \cdot M_B \cdot S
  \]
Bicubic Bezier Patch

Using data array $P^{ij}$

$$p(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} b_i(u)b_j(v)\vec{p}_{ij} = u^TPM_{ij}v$$

Cubic Bezier Blending Functions

$$b(u) = \begin{cases} 0 & u < 0 \\ \frac{(1-u)^3}{3u(1-u)} & 0 \leq u < 1 \\ 0 & 1 \leq u \end{cases}$$

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

Bicubic Bézier Patches

• Expanding the summation

$$p(u,v) = \sum_{i=0}^{1} \sum_{j=0}^{1} b_i(u)b_j(v)\vec{p}_{ij} = b_0(u)b_0(v)\vec{p}_{00} + b_1(u)b_0(v)\vec{p}_{01} + b_0(u)b_1(v)\vec{p}_{02} + b_1(u)b_1(v)\vec{p}_{03} + \text{etc.}$$

Features of Bicubic Bezier Patch

• Interpolates 4 corner control points
• 4 edges are Bezier curves
• Lies within convex hull of control points
• Normal at 4 corners from nearby CPs

Plotting Isolines

Faceting Animation
Faceting Overview

- Double loop that increments through the u and v parameters
  - Values between 0 and 1
- For each (u,v) pair calculate 3D point on patch. Keep track of linear index.
- This produces a 2-D array of 3D points on the patch and their indices to the linear array
- Define triangles that tessellate the patch

Defining the Triangles

```c
// This assumes that indices to the vertices are
// in a 2D array, verts(i,j)

num_tri = 0
for i = 0 to (num_u - 2)
  for j = 0 to (num_v - 2)
    triangles[num_tri++] = (verts[i,j], verts[i+1,j], verts[i+1,j+1])
    triangles[num_tri++] = (verts[i,j], verts[i+1,j+1], verts[i,j+1])
```

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

Bezier Surface: Example

- Increased facet resolution
- Rendered
B-spline Surfaces

\[ x(s,t) = T^T \cdot M_{B_s} \cdot G_{B_s} \cdot M_{B_t} \cdot S \]
\[ y(s,t) = T^T \cdot M_{B_s} \cdot G_{B_s} \cdot M_{B_t} \cdot S \]
\[ z(s,t) = T^T \cdot M_{B_s} \cdot G_{B_s} \cdot M_{B_t} \cdot S \]

- 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:

\[
\frac{\delta}{\delta s} Q(s,t) = \frac{\delta}{\delta s} \left( T^T \cdot M^T \cdot G \cdot M \cdot S \right)
\]
\[ = T^T \cdot M^T \cdot G \cdot M \cdot \frac{\delta}{\delta s} (S)
\]
\[ = T^T \cdot M^T \cdot G \cdot M \cdot [3s^2 \ 2s \ 1 \ 0]
\]

- Next, compute the \( t \) tangent vector:

\[
\frac{\delta}{\delta t} Q(s,t) = \frac{\delta}{\delta t} \left( T^T \cdot M^T \cdot G \cdot M \cdot S \right)
\]
\[ = \frac{\delta}{\delta t} (T^T) \cdot M^T \cdot G \cdot M \cdot S
\]
\[ = [3t^2 \ 2t \ 1 \ 0]^T \cdot M^T \cdot G \cdot M \cdot S
\]

Computing the Normals to Surfaces

- Since \( s \) and \( t \) are tangent to the surface, their cross product is the normal vector to the surface!

\[
\frac{\delta}{\delta s} Q(s,t) \times \frac{\delta}{\delta t} Q(s,t) = \left[ y_s z_t - y_t z_s, \ z_s x_t - z_t x_s, \ x_s y_t - x_t y_s \right]
\]

\( x_s \) - \( x \) component of \( s \) tangent
\( y_s \) - \( y \) component of \( s \) tangent
\( z_s \) - \( z \) component of \( s \) tangent

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
  - Draw the curve
    - De Boor, D'Castejeau
  - 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
    - Iterative evaluation
    - 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
  - Screen flatness of silhouette edges
    - 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!
  \[ N(X) \cdot L = 0 \]
  - Where \( N \) is the normal at the point where \( L \), the line of sight, hits the surface

Silhouette Determination

\[ N \cdot L = 0 \]

Programming Assignment 3

- Process command line arguments
- Read in control points from file
- Double loop through \( u \) & \( v \) parameters
- For each \((u,v)\) pair compute 3D point on Bezier patch
- Once you’ve computed the 3D points, define the triangles that connect them
- If shading, compute exact normals at each mesh vertex
- Output all data as Open Inventor