Overview
• 3D model representations
• Mesh formats
• Bicubic surfaces
• Bezier surfaces
• Normals to surfaces
• Direct surface rendering

Representing 3D Objects
• Exact
  – Wireframe
  – Parametric Surface
  – Solid Model
    • CSG
    • BRep
    • Implicit Solid Modeling
• Approximate
  – Face / 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

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), BReps and feature-based

Representing 3D Objects
• Exact
  – Precise model of object topology
  – Mathematically represent all geometry
• Approximate
  – A discretization of the 3D object
  – Use simple primitives to model topology and geometry
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
    - Loss of interpolation and guess work

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/](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
  - X,Y,Z point
  - Pointers to end-point vertices
  - Pointers to n coincident edges
  - Pointers to adjacent faces
  - Pointer to next edge
  - Pointer to previous edge

Biparametric Surfaces
- Biparametric surfaces
  - A generalization of parametric curves
  - 2 parameters: x, 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 incrementing \( t_1 \) to \( t_2 \), we get a new curve
- The combination of these curves is a surface
- \( G(t) \) are 3D curves

Bicubic Surfaces

- Each \( G(t) \) is \( G(t) = G_s \cdot M \cdot T \), where

\[
G_s = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \end{bmatrix}
\]
- Transposing \( G(t) \), we get

\[
G(t) = T^T \cdot M^T \cdot G_s^T
\]

\[
= T^T \cdot M^T \cdot \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \end{bmatrix}^T
\]

Bicubic Surfaces

- Substituting \( G(t) \) into \( Q(s) = G \cdot M \cdot S \), we get \( Q(s, t) \)
- The \( g_{ij} \), etc. are the control points for the Bicubic surface patch:

\[
Q(s, t) = T^T \cdot M^T \cdot \begin{bmatrix} s_{11} & s_{21} & s_{31} & s_{41} \\
                    s_{12} & s_{22} & s_{32} & s_{42} \\
                    s_{13} & s_{23} & s_{33} & s_{43} \\
                    s_{14} & s_{24} & s_{34} & s_{44} \end{bmatrix} \cdot M \cdot S
\]

Bicubic Surfaces

- Writing out 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 Bézier Patch

- Bézier Surfaces (similar definition)

\[
x(s, t) = T^T \cdot M^T \cdot G_B \cdot M_B \cdot S
\]

\[
y(s, t) = T^T \cdot M^T \cdot G_B \cdot M_B \cdot S
\]

\[
z(s, t) = T^T \cdot M^T \cdot G_B \cdot M_B \cdot S
\]
Bicubic Bezier Patch

Using data array \( P = \{p_{ij}\} \)

\[
\hat{p}(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} p_{ij} b_i(u) b_j(v) u^{3-i} v^{3-j}
\]

\( \rho_p(\hat{p}(u,v)) = i \rho_p(u) + j \rho_p(v) + \text{etc.} \)

\( 0 \leq u, v \leq 1 \)

Cubic Bezier Blending Functions

\[
b(u) = \begin{cases} 
1 - u)^3 & \text{if } u \leq 1 \\
3(1-u)^2u & \text{if } 1 < u \leq 2 \\
3u^2(1-u) & \text{if } 2 < u \leq 3 \\
u^3 & \text{if } 3 < u 
\end{cases}
\]

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

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

Bezier Patch Matrix Form

\[
P(u,v) = u^T M_B P M_B^T v
\]

\[
= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 1 & 0 & 0 & 2 \\ -1 & -3 & 3 & -1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}
\]

Plotting Isolines
**Faceting Animation**

- 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**

```cpp
// 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

47

Bezier Surface: Example

- Increased facet resolution
- Rendered

48

B-spline Surfaces

\[
x(s,t) = T^T \cdot M^b_{st} \cdot G_{st} \cdot M_{st} \cdot S
\]
\[
y(s,t) = T^T \cdot M^b_{st} \cdot G_{st} \cdot M_{st} \cdot S
\]
\[
z(s,t) = T^T \cdot M^b_{st} \cdot G_{st} \cdot M_{st} \cdot S
\]
- Representation for B-spline patches
- $C^1$ continuity across boundaries is automatic with B-splines

49

Normals to Surfaces

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

50

Computing the Normals to Surfaces

- For a bicubic surface, first, compute the $s$ tangent vector

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

51

Computing the Normals to Surfaces

- Next, compute the $t$ tangent vector:

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

52
Computing the Normals to Surfaces

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

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

- $x_s$ - $x$ component of $s$ tangent
- $y_s$ - $y$ component of $s$ tangent
- $z_s$ - $z$ 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
  - $\theta$ of angle of rotation
- Examples: cylinder, cone, sphere, torus

Surface of Revolution

- Directrix:
  - $D(u) = (f(u), 0, g(u))$
- Surface:
  - $S(u, \theta) = (f(u)\cos(\theta), f(u)\sin(\theta), g(u))$
  - $0 \leq u \leq 1, 0 \leq \theta \leq 2\pi$
- Tangents:
  - $\frac{\partial S}{\partial u} = (f'(u)\cos(\theta), f'(u)\sin(\theta), g'(u))$
  - $\frac{\partial S}{\partial \theta} = (-f(u)\sin(\theta), f(u)\cos(\theta), 0)$
  - $N(u, \theta) = \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial \theta}$

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' Casteljeau
- 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

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