Curved Surfaces

• Motivation
  - Exact boundary representation for some objects
  - More concise representation than polygonal mesh

Parametric Surfaces

• Boundary defined by parametric functions:
  - \( x = f_x(u,v) \)
  - \( y = f_y(u,v) \)
  - \( z = f_z(u,v) \)

- Example: ellipsoid
  - \( x = r_x \cos \phi \cos \theta \)
  - \( y = r_y \cos \phi \sin \theta \)
  - \( z = r_z \sin \phi \)

Surface of revolution

• Idea: take a curve and rotate it about an axis

Swept surface

- Idea: sweep one curve along path of another curve
Parametric Surfaces

Advantage: easy to enumerate points on surface.
Disadvantage: need piecewise-parametric surface to describe complex shape.

Piecewise Parametric Surfaces

Surface is partitioned into parametric patches:

Same ideas as parametric splines!

Parametric Patches

• Each patch is defined by blending control points

Same ideas as parametric curves!

Let the Control Points Move

• Call the Bézier parameter \( v \), and let the \( N+1 \) control points depend on some other parameter \( u \):

\[
p(v, u) = \sum_{i=0}^{N} p_i(u)B_i^N(v)
\]

• Each “\( u \)-contour” is a normal Bézier curve, but at different \( u \) values, the control points are at different positions

• Think of the surface as a changing Bézier curve sweeping through space

• How do the control points change?

Bézier Patches

• Let’s allow the control points to move along their own Bézier curves:

\[
p_i(u) = \sum_{j=0}^{N} p_{i,j} B_j^N(u)
\]

• Putting this together with our original definition of our surface, we get the tensor product form for the Bézier patch:

\[
p(v, u) = \sum_{i=0}^{N} \sum_{j=0}^{N} p_i B_j^N(v) B_i^N(u)
\]
Parametric Bicubic Patches

Point \( Q(u,v) \) on any patch is defined by combining control points with polynomial blending functions:

\[
Q(u,v) = \mathbf{U} \mathbf{M} \mathbf{V}^T
\]

\[
\mathbf{U} = \begin{bmatrix} u & u^2 & u^3 & 1 \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} v & v^2 & v^3 & 1 \end{bmatrix}
\]

Where \( \mathbf{M} \) is a matrix describing the blending functions for a parametric cubic curve (e.g., Bezier, B-spline, etc.)

Bez

\[
\mathbf{M}_{t_{ij}} = \begin{bmatrix}
\frac{1}{6} & \frac{1}{2} & \frac{1}{6} & 0 \\
\frac{1}{6} & -\frac{1}{2} & \frac{1}{6} & 0 \\
\frac{1}{6} & 0 & \frac{1}{6} & 0 \\
\frac{1}{6} & \frac{1}{2} & \frac{1}{6} & 0
\end{bmatrix}
\]

Bez

\[
\mathbf{M}_{t_{ijk}} = \begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

Bez

• Properties:
  • Interpolates four corner points
  • Convex hull
  • Local control

Bez

• \( C^0 \) continuity requires aligning boundary curves
Bezier Surfaces

- $C^1$ continuity requires aligning boundary curves and derivatives (a reason to prefer subdiv. surf.)

Drawing Bezier Surfaces

- Simple approach is to loop through uniformly spaced increments of $u$ and $v$

```cpp
DrawSurface(void)
{
    for (int i = 0; i < imax; i++) {
        float u = umin + i * ustep;
        for (int j = 0; j < jmax; j++) {
            float v = vmin + j * vstep;
            DrawQuadrilateral(...);
        }
    }
}
```

Bézier Curves in OpenGL

- Use evaluators
- glMap defines the set of control points
- glMapGrid defines how finely to evaluate the surface
- glEvalCoord/glEvalMesh cause the mesh to be drawn

Defining the Control Points

- glMap2 [df](target, u1, u2, ustride, uorder, v1, v2, vstride, vorder, points)
  - target specifies what OpenGL command will be executed when this mesh is evaluated, and what's in the control mesh. For drawing, usually use GL_MAP2_VERTEX3
  - u1,u2,v1,v2 define a mapping from values passed to glEvalCoord to (0,1), the domain of the Bézier functions
  - ustride, vstride indicate how the data is packed in the array
  - uorder, vorder define the dimensions of the point array
  - points is the actual data
- glMap2d(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12, 4, &ctrlpoints[0][0][0]);

Defining the Mesh Parameters

- glMapGrid specifies how the mesh will be evaluated based on the control points
- glMapGrid2 [df](un, u1, u2, vn, v1, v2)
  - un, vn define the number of partitions at which to evaluate the surface
  - u1,u2, v1,v2 define the range of grid variables
- glMapGrid2 d(20, 0.0, 1.0, 20, 0.0, 1.0);
Drawing the Mesh

- We can draw the whole mesh at once with `gLErrorMesh`
  - `gLErrorMesh2(mode, i1, i2, j1, j2)`
    - `mode` specifies points, lines, or polygons
    - `i1, i2, j1, j2` define the range over which to evaluate the mesh
  - `gLErrorMesh2(GL_LINE, 0, 20, 0, 20);`

Drawing Bezier Surfaces

- One problem with adaptive subdivision is avoiding cracks at boundaries between patches at different subdivision levels

Avoid these cracks by adding extra vertices and triangulating quadrilaterals whose neighbors are subdivided to a finer level.

Parametric Surfaces

- Advantages:
  - Easy to enumerate points on surface
  - Possible to describe complex shapes
- Disadvantages:
  - Control mesh must be quadrilaterals
  - Continuity constraints difficult to maintain
  - Hard to find intersections

Next Time

- Subdivision Surfaces

Blender (www.blender.nl)