Bézier Surfaces

Beziers Surfaces Introduction

• Constructing a surface relies very much on the ideas behind constructing curves
• Surfaces can be thought of as ‘Bézier curves in all directions’ across the surface
• Tensor products of Bézier curves
• Teapot most famous example
  – produced entirely by Bézier surfaces

Tensor Product

• Of two vectors:
  \[
  \begin{bmatrix}
  a_1 & a_2 & a_3 \\
  b_1 & b_2 & b_3 \\
  \end{bmatrix}
  \begin{bmatrix}
  h_1 & h_2 & h_3 \\
  \end{bmatrix}
  =
  \begin{bmatrix}
  a_1h_1 & a_2h_1 & a_3h_1 \\
  a_1h_2 & a_2h_2 & a_3h_2 \\
  a_1h_3 & a_2h_3 & a_3h_3 \\
  \end{bmatrix}
  \]

• Similarly, we can define a surface as the tensor product of two curves....
Bilinear Patch

Bi-lerp a (typically non-planar) quadrilateral

Notation: $L(P_1, P_2, \alpha) \equiv (1 - \alpha)P_1 + \alpha P_2$

$Q(s, t) = L(L(P_1, P_2, s), L(P_3, P_4, t), s)$

Bicubic Bezier Patch

Notation: $CB(P_1, P_2, P_3, P_4, \alpha)$ is Bezier curve with control points $P_i$ evaluated at $\alpha$

Define "Tensor-product" Bezier surface

$Q(s, t) = CB(CB(P_1, P_2, P_3, P_4, s), CB(P_5, P_6, P_7, P_8, t), CB(P_9, P_{10}, P_{11}, P_{12}, s))$

Editing Bicubic Bezier Patches

Curve Basis Functions

Surface Basis Functions
Control Points

- Consider the \((m+1) \times (n+1)\) array of 3D control points.
- This array can be used to define a Bezier of surface of degree \(m\) and \(n\).
- If \(m=n=3\) this is called ‘bi-cubic’.
- The same relation between surface and control points holds as in curves:
  - If the points are on a plane the surface is a plane.
  - If the edges are straight the Bezier surface edges are straight.
  - The entire surface lies inside the convex hull of the control points.

Rendering – de Casteljau

- Use de Casteljau to subdivide each row.
- Then use de Casteljau to subdivide each of the 7 resulting columns.
- This will result in 4 sets of \((m+1) \times (n+1)\) array with one common row and one common column.
- If all points are on a plane and the edges are straight line then we get a polygon with 4 vertices.
- So the recursive algorithm is as follows:

Subdivision 4x4 Cubic Case

This gives 4 sets of 4x4 arrays of control points. In each case the middle values are shared by the two adjacent sets.
Rendering 3D de Casteljau

```c
typedef struct{
    float x, y, z;
} Point3D;

typedef Point3D ControlPointArray[10];

void Bicubic3D(ControlPointArray plane)
{
    ControlPointArray x, y, z;
    // Calculation logic here
    // Final result
}
```

Testing Colinearity

- This is where the computational work of the algorithm is located.
- If the equation of the plane is ax+by+cz=d and (x,y,z) is a point then its distance from the plane is:

  \[ D = \frac{(ax+by+cz-d)^2}{\sqrt{a^2+b^2+c^2}} \]

- So we have an analogy to the curve case, except also we should check that the edges are straight, and that adjacent regions have been split to the same level.
- A simple approach is to just run the recursion to the same level irrespective of testing whether the final pieces are really flat.

Modeling with Bicubic Bezier Patches

- Original Teapot specified with Bezier Patches
Bernstein Basis Representation

- Mathematically the surface can be represented as:

\[ F(r, u) = \sum_{j=0}^{m} \sum_{i=0}^{n} B_{i,j}(r) B_{j,i}(u) \]

- Note that it is easy to see that this is simply a ‘Bezier curve of Bezier curves’.

B-Spline Surfaces

- These can be constructed in exactly the same way, except that there will be a knot sequence for the rows and the columns.

  \[ t_1, t_2, \ldots, t_{m+3} \text{ for each column, and} \]

  \[ u_1, u_2, \ldots, u_{n+3} \text{ for each row.} \]

- The simplest approach is on a row by row and column by column basis convert the B-Spline control points to Bezier control points and then render the Bezier surfaces.

Conclusions

- Surfaces are a simple extension to curves
- Really just a tensor-product between two curves
  - One curve gets extruded along the other