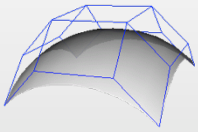
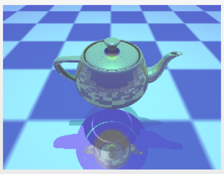
Bézier Surfaces



http://www.ibiblio.org/e-notes/Splines/fig/surf2.gif

Bezier Surfaces Introduction

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

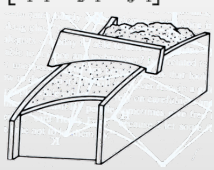


Tensor Product

- Of two vectors:

$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \otimes \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1b_1 & a_2b_1 & a_3b_1 \\ a_1b_2 & a_2b_2 & a_3b_2 \\ a_1b_3 & a_2b_3 & a_3b_3 \\ a_1b_4 & a_2b_4 & a_3b_4 \end{bmatrix}$$

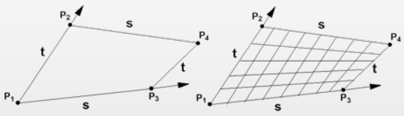
- Similarly, we can define a surface as the tensor product of two curves....



Farin, Curves and Surfaces for Computer Aided Geometric Design

Bilinear Patch

Bi-lerp a (typically non-planar) quadrilateral



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

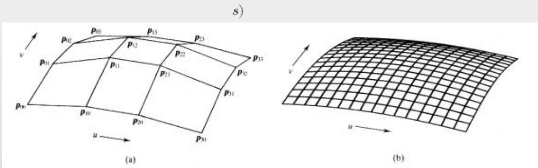
$$Q(s,t) = \mathbf{L}(\mathbf{L}(P_1, P_2, t), L(P_3, P_4, t), s)$$

Bicubic Bezier Patch

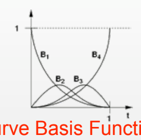
Notation: $\mathbf{CB}(P_1, P_2, P_3, P_4, \alpha)$ is Bézier curve with control points P_i evaluated at α

Define "Tensor-product" Bézier surface

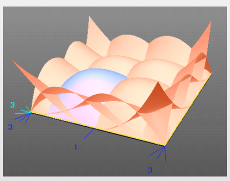
$$Q(s,t) = \mathbf{CB}(\ \mathbf{CB}(P_{00}, P_{01}, P_{02}, P_{03}, t), \mathbf{CB}(P_{10}, P_{11}, P_{12}, P_{13}, t), \mathbf{CB}(P_{20}, P_{21}, P_{22}, P_{23}, t), \mathbf{CB}(P_{30}, P_{31}, P_{32}, P_{33}, t), s)$$



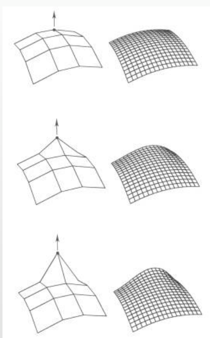
Editing Bicubic Bezier Patches



Curve Basis Functions



Surface Basis Functions



Control Points

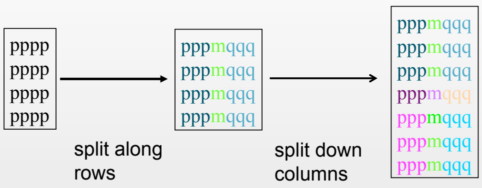
- Consider the (m+1)*(n+1) array of 3D control points
- This array can be used to define a Bezier of surface of degree m and n.
- If m=n=3 this is called ‘bi-cubic’.
- The same relation between surface and control points holds as in curves
  - If the points are on a plane the surface is a plane
  - If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

$$\begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0n} \\ p_{10} & p_{11} & \cdots & p_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ p_{m0} & p_{m1} & \cdots & p_{mn} \end{bmatrix}$$

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)*(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 4*4 Cubic Case



This gives 4 sets of 4*4 arrays of control points. In each case the middle values are shared by the two adjacent sets.

Rendering 3D de Casteljau

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

typedef Point3D ControlPointArray[4][4];

void Bezier3D(ControlPointArray p) {
        ControlPointArray q,r,s,t;
        if(Coplanar(p)) RenderPolygon(p[0][0],p[3][0],p[3][3],p[0][3]);
        else{
                /*split p into q,r,s,t*/
                Split3D(p,q,r,s,t);
                Bezier3D(q);
                Bezier3D(r);
                Bezier3D(s);
                Bezier3D(t);
        }
}
```

Testing Colinearity

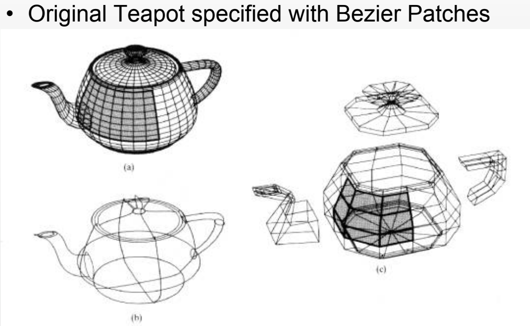
- This is where the computational work of the algorithm is located.
- If the equation of the plane is ax+by+cz=d and (x,y,z) is a point then its distance from the plane is:

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

- 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:

$$F(t, u) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_{m,i}(t) p_{ij} B_{n,j}(u)$$
$$t, u \in [0, 1]$$

- Note that it is easy to see that this is simple 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 for the colums.

$t_1, t_2, \ldots, t_{m+3}$ for each column, and

$u_1, u_2, \ldots, u_{n+3}$ 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