

CS770/870 Spring 2017

Surface Generation

- Primary resources used in preparing these notes:

Foley, van Dam, Feiner, Hughes, Phillips, **Introduction to Computer Graphics**, Addison-Wesley, 1993.

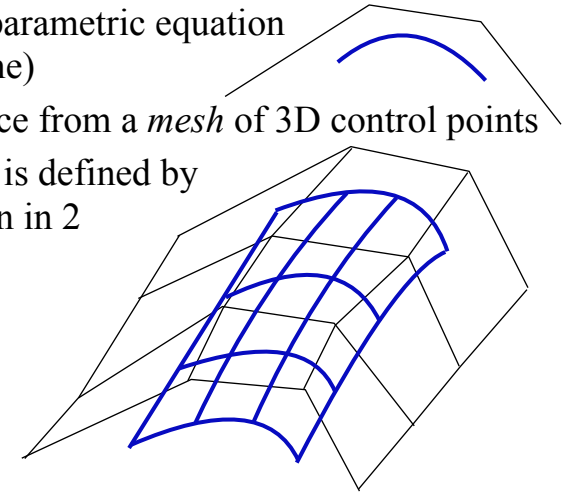
04/24/2017

CS770/870 Spring 2017 Bergeron

1

Overview

- Can define 3D curve from (nonplanar) 3D control points
 - Curve defined by parametric equation in t (here, a B-spline)
- Can define a 2D surface from a *mesh* of 3D control points
 - Each surface point is defined by parametric equation in 2 variables (s, t)



CS770/870 Spring 2017 Bergeron

2

Parametric Bicubic Surfaces

- Generalize parametric bicubic curves
 - curve(s) = PMS where M is blending function coefficient matrix, $P = [P_0, P_1, P_2, P_3]$ and $S = \begin{bmatrix} 1 \\ s \\ s^2 \\ s^3 \end{bmatrix}$
- Suppose now that each point in P is a function of t :
 - surface(s, t) = $[P_0(t), P_1(t), P_2(t), P_3(t)]MS$
 - for $t=t_1$, $S(s, t_1)$ is a curve because $P(t_1)$ is constant
 - Let $t_2=t_1+dt$; $P(t_2)$ is also a curve that is similar to $P(t_1)$ if dt is small; $P(t_2)$ is similar to $P(t_2+dt)$, etc.
- If $P_i(t)$ are cubics, $P_i(t) = P_iMT$, where $P_i = [p_{i0} \ p_{i1} \ p_{i2} \ p_{i3}]$
 - $(P_iMT)^T = T^T M^T P_i^T$ and $S(s, t) = T^T M^T PMS$ where $P = [p_{ij}]$ for $i, j = 0, 1, 2, 3$

CS770/870 Spring 2017 Bergeron

3

B-Spline Surface

- B-spline patches:
 - $B(s, t) = T^T M^T_B P M_B S$ where $S = [1, s, s^2, s^3]^T$ (column vector)
 $T^T = [1, t, t^2, t^3]$ (row vector)

$$M_B = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \quad P = \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix}$$

- Each $p_{ij} = [x_{ij}, y_{ij}, z_{ij}]$, so
 $X(s, t) = T^T M^T_B X M_B S$, where X = array of x_{ij} values
 $Y(s, t) = T^T M^T_B Y M_B S$, where Y = array of y_{ij} values
 $Z(s, t) = T^T M^T_B Z M_B S$, where Z = array of z_{ij} values

CS770/870 Spring 2017 Bergeron

4

Surface Normals

- Normals to parametric bicubic surfaces are easy:

- First get tangents at a point on surface

$$\partial B(s,t)/\partial s = \partial/\partial s (T^T M^T B P M_B S)$$

$$= T^T M^T B P M_B \partial S/\partial s$$

$$= T^T M^T B P M_B \partial/\partial s [1, s, s^2, s^3]$$

$$= T^T M^T B P M_B [0, 1, 2s, 3s^2]$$

$$\partial B(s,t)/\partial t = \partial/\partial t (T^T M^T B P M_B S)$$

$$= [0, 1, 2t, 3t^2] M^T B P M_B S$$

- Then take cross product

$$\partial B/\partial s \times \partial B/\partial t = [y_s z_t - y_t z_s, z_s x_t - z_t x_s, x_s y_t - x_t y_s]$$

Line Display of Bicubic Surfaces

- Curved line display is straightforward (from Foley et al.)

```
// X(s,t), Y(s,t), Z(s,t) implement M^T B P M_B products
void drawSurf( float[4][4][3] P, int ns, int nt, int n)
{
    dCurve = 1/n // step size for curve drawing
    dS = 1/ns    // s parameter step size
    dT = 1/nt    // t parameter step size
    for ( i = 0; i < ns; i++ ) // curves with constant s
    {
        s = i * dS
        move( X(s,0), Y(s,0), Z(s,0) )
        for ( j=0; j<n; j++ ) //draw curve in s dir
        {
            t = j * dT
            line( X(s,t), Y(s,t), Z(s,t) )
        }
    }
    for ( i = 0; i < nt; i++ ) // curves with constant t
    {
        t = i * dT
        move( X(0,t), Y(0,t), Z(0,t) )
        for ( j=0; j<n; j++ ) //draw curve in s dir
        {
            s = j * dS
            line( X(s,t), Y(s,t), Z(s,t) )
        }
    }
}
```

Matrix approach

- For simplicity of presentation Foley et al. implement the X, Y, and Z functions independently as algebraic expressions, which means re-evaluating blending function calculations.
- Can use a matrix approach and have more shared code:

```
for each B-spline "patch"
    build 4x4 P_x=M^T_B X M_B, P_y=M^T_B Y M_B, and P_z=M^T_B Z M_B
    for each s in [0,1] by ds
        build S matrix
        for each t in [0,1] by dt
            build T matrix
            X(s,t)=T^T P_x S; Y(s,t)=T^T P_y S; Z(s,t)=T^T P_z S;
```

Triangle Generation

- For surface (vs. line) drawing, often have larger delta
- Line drawing or matrix approach can get 2D point array
 - Triangulate between neighbors
- Modify either algorithm to save previous point in inner *for* loop and entire “row” of points in previous row, so can generate triangles as generating points.
- GLSL Geometry Shader is designed specifically for surface generation of this sort.
 - Geometry shader inserts new vertices into the shader pipeline
 - Matrix approach is natural and should be more efficient.