

## Splines

Cubic B-Splines  
Nonuniform Rational B-Splines  
Rendering by Subdivision  
Curves and Surfaces in OpenGL  
[Angel, Ch 10.7-10.14]

February 21, 2002  
Frank Pfenning  
Carnegie Mellon University

<http://www.cs.cmu.edu/~fp/courses/graphics/>

## Review

- Cubic polynomial form for curve  

$$p(u) = c_0 + c_1u + c_2u^2 + c_3u^3 = \sum_{k=0}^3 c_k u^k$$
- Each  $c_k$  is a column vector  $[c_{kx} \ c_{ky} \ c_{kz}]^T$
- Solve for  $c_k$  given control points
- Interpolation: 4 points
- Hermite curves: 2 endpoints, 2 tangents
- Bezier curves: 2 endpoints, 2 tangent points

02/21/2002

15-462 Graphics I

2

## Splines

- Approximating more control points



- $C^0$  continuity: points match
- $C^1$  continuity: tangents (derivatives) match
- $C^2$  continuity: curvature matches
- With Bezier segments or patches:  $C^0$

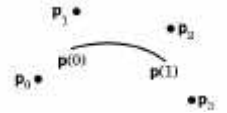
02/21/2002

15-462 Graphics I

3

## B-Splines

- Use 4 points, but approximate only middle two



- Draw curve with overlapping segments  
0-1-2-3, 1-2-3-4, 2-3-4-5, 3-4-5-6, etc.
- Curve may miss all control points
- Smoother at joint points

02/21/2002

15-462 Graphics I

4

## Cubic B-Splines

- Need  $m+2$  control points for  $m$  cubic segments
- Computationally 3 times more expensive
- $C^2$  continuous at each interior point
- Derive as follows:
  - Consider two overlapping segments
  - Enforce  $C^0$  and  $C^1$  continuity
  - Employ symmetry
  - $C^2$  continuity follows

02/21/2002

15-462 Graphics I

5

## Deriving B-Splines

- Consider points
  - $p_{i-2}, p_{i-1}, p_i, p_{i+1}$
  - $p(0)$  approx  $p_{i-1}, p(1)$  approx  $p_i$
  - $q(0)$  approx  $p_{i-2}, q(1)$  approx  $p_{i-1}$
- Condition 1:  $p(0) = q(1)$ 
  - Symmetry:  $p(0) = q(1) = 1/6(p_{i-2} + 4p_{i-1} + p_i)$
- Condition 2:  $p'(0) = q'(1)$ 
  - Geometry:  $p'(0) = q'(1) = 1/2((p_i - p_{i-1}) + (p_{i-1} - p_{i-2})) = 1/2(p_i - p_{i-2})$

02/21/2002

15-462 Graphics I

6

## B-Spline Geometry Matrix

- Conditions at  $u = 0$ 
  - $p(0) = c_0 = 1/6 (p_{i-2} + 4p_{i-1} + p_i)$
  - $p'(0) = c_1 = 1/2 (p_i - p_{i-2})$
- Conditions at  $u = 1$ 
  - $p(1) = c_0 + c_1 + c_2 + c_3 = 1/6 (p_{i-1} + 4p_i + p_{i+1})$
  - $p'(1) = c_1 + 2c_2 + 3c_3 = 1/2 (p_{i+1} - p_{i-1})$

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = M_S \begin{bmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \end{bmatrix}, M_S = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

02/21/2002

15-462 Graphics I

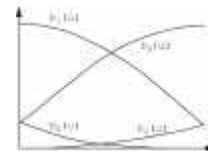
7

## Blending Functions

- Calculate cubic blending polynomials

$$b(u) = M_S^T u = \frac{1}{6} \begin{bmatrix} (1-u)^3 \\ 4-6u^2+3u^3 \\ 1+3u+3u^2-3u^3 \\ u^3 \end{bmatrix}$$

- Note symmetries



02/21/2002

15-462 Graphics I

8

## Convex Hull

- For  $0 \leq u \leq 1$ , have  $0 \leq b_k(u) \leq 1$
- Recall:
 
$$p(u) = b_{i-2}(u)p_{i-2} + b_{i-1}(u)p_{i-1} + b_i(u)p_i + b_{i+1}(u)p_{i+1}$$
- So each point  $p(u)$  lies in convex hull of  $p_k$

02/21/2002

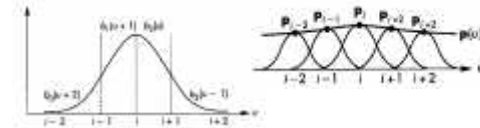
15-462 Graphics I

9

## Spline Basis Functions

- Total contribution  $B_i(u)p_i$  of  $p_i$  is given by

$$B_i(u) = \begin{cases} 0 & u < i-2 \\ b_0(u+2) & i-2 \leq u < i-1 \\ b_1(u+1) & i-1 \leq u \leq i \\ b_2(u) & i \leq u < i+1 \\ b_3(u-1) & i+1 \leq u < i+2 \\ 0 & i-2 \leq u \end{cases}$$



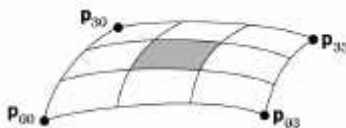
02/21/2002

15-462 Graphics I

10

## Spline Surface

- As for Bezier patches, use 16 control points
- Start with blending functions
 
$$p(u, v) = \sum_{i=0}^3 \sum_{k=0}^3 b_i(u)b_k(v)p_{ik}$$
- Need 9 times as many splines as for Bezier



02/21/2002

15-462 Graphics I

11

## Assessment: Cubic B-Splines

- More expensive than Bezier curves or patches
- Smoother at join points
- Local control
  - How far away does a point change propagate?
- Contained in convex hull of control points
- Preserved under affine transformation
- How to deal with endpoints?
  - Closed curves (uniform periodic B-splines)
  - Non-uniform B-Splines (multiplicities of knots)

02/21/2002

15-462 Graphics I

12

## General B-Splines

- Generalize from cubic to arbitrary order
- Generalize to different basis functions
- Read: [Angel, Ch 10.8]
- Knot sequence  $u_{\min} = u_0 \leq \dots \leq u_n = u_{\max}$
- Repeated points have higher "gravity"
- Multiplicity 4 means point must be interpolated
- $\{0, 0, 0, 0, 1, 2, \dots, n-1, n, n, n, n\}$  solves boundary problem

02/21/2002

15-462 Graphics I

13

## Nonuniform Rational B-Splines (NURBS)

- Exploit homogeneous coordinates

$$p_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \simeq w_i \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = q_i$$

- Use perspective division to renormalize

$$p(u) = \frac{\sum_{i=0}^n B_i(u) w_i p_i}{\sum_{i=0}^n B_i(u) w_i}$$

- Each component of  $p(u)$  is rational function of  $u$
- Points not necessarily uniform (NURBS)

02/21/2002

15-462 Graphics I

14

## NURBS Assessment

- Convex-hull and continuity props. of B-splines
- Preserved under perspective transformations
  - Curve with transformed points = transformed curve
- Widely used (including OpenGL)

02/21/2002

15-462 Graphics I

15

## Outline

- Cubic B-Splines
- Nonuniform Rational B-Splines (NURBS)
- Rendering by Subdivision
- Curves and Surfaces in OpenGL

02/21/2002

15-462 Graphics I

16

## Rendering by Subdivision

- Divide the curve into smaller subpieces
- Stop when "flat" or at fixed depth
- How do we calculate the sub-curves?
  - Bezier curves and surfaces: easy (next)
  - Other curves: convert to Bezier!

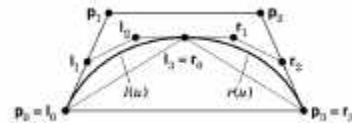
02/21/2002

15-462 Graphics I

17

## Subdividing Bezier Curves

- Given Bezier curve by  $p_0, p_1, p_2, p_3$
- Find  $l_0, l_1, l_2, l_3$  and  $r_0, r_1, r_2, r_3$
- Subcurves should stay the same!



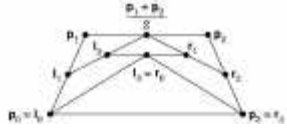
02/21/2002

15-462 Graphics I

18

## Construction of Bezier Subdivision

- Use algebraic reasoning



- $l(0) = l_0 = p_0$
- $l(1) = l_3 = p(1/2) = 1/8(p_0 + 3p_1 + 3p_2 + p_3)$
- $l'(0) = 3(l_1 - l_0) = p'(0) = 3/2(p_1 - p_0)$
- $l'(1) = 3(l_3 - l_2) = p'(1/2) = 3/8(-p_0 - p_1 + p_2 + p_3)$
- Note parameter substitution  $v = 2u$  so  $dv = 2du$

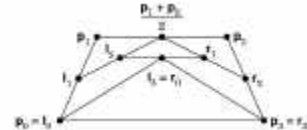
02/21/2002

15-462 Graphics I

19

## Geometric Bezier Subdivision

- Can also calculate geometrically



- $l_1 = 1/2(p_0 + p_1)$ ,  $r_2 = 1/2(p_2 + p_3)$
- $l_2 = 1/2(l_1 + 1/2(p_1 + p_2))$ ,  $r_1 = 1/2(r_2 + 1/2(p_1 + p_2))$
- $l_3 = r_0 = 1/2(l_2 + r_1)$ ,  $l_0 = p_0$ ,  $r_3 = p_3$

02/21/2002

15-462 Graphics I

20

## Recall: Bezier Curves

- Recall  $u^T = [1 \ u \ u^2 \ u^3]$
- Express  $p(u) = c_0 + c_1u + c_2u^2 + c_3u^3$

$$= u^T \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = u^T M_B \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

$$M_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & -1 \end{bmatrix}$$

02/21/2002

15-462 Graphics I

21

## Subdividing Other Curves

- Calculations more complex
- Trick: transform control points to obtain identical curve as Bezier curve!
- Then subdivide the resulting Bezier curve
- Bezier:  $p(u) = u^T M_b p$
- Other curve:  $p(u) = u^T M q$ ,  $M$  geometry matrix
- Solve:  $q = M^{-1} M_b p$  with  $p = M_b^{-1} M q$

02/21/2002

15-462 Graphics I

22

## Example Conversion

- From cubic B-splines to Bezier:

$$M_B^{-1} M_S = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{bmatrix}$$

- Calculate Bezier points  $p$  from  $q$
- Subdivide as Bezier curve

02/21/2002

15-462 Graphics I

23

## Subdivision of Bezier Surfaces

- Slightly more complicated
- Need to calculate interior point
- Cracks may show with uneven subdivision
- See [Angel, Ch 10.9.4]

02/21/2002

15-462 Graphics I

24

## Outline

- Cubic B-Splines
- Nonuniform Rational B-Splines (NURBS)
- Rendering by Subdivision
- Curves and Surfaces in OpenGL

02/21/2002

15-462 Graphics I

25

## Curves and Surface in OpenGL

- Central mechanism is evaluator
- Defined by array of control points
- Evaluate coordinates at  $u$  (or  $u$  and  $v$ ) to generate vertex
- Define Bezier curve:  $type = GL\_MAP\_VERTEX\_3$   
`glMap1f(type, u0, u1, stride, order, point_array)`
- Enable evaluator  
`glEnable(type)`
- Evaluate Bezier curve  
`glEvalCoord1f(u)`

02/21/2002

15-462 Graphics I

26

## Example: Drawing a Bezier Curve

- 4 control points

```
GLfloat ctrlpnts[4][3] = {  
    {-4.0, -4.0, 0.0}, {-2.0, 4.0, 0.0},  
    {2.0, -4.0, 0.0}, {4.0, 4.0, 0.0}};
```

- Initialize

```
void init()  
{  
    ...  
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4,  
            &ctrlpnts[0][0]);  
    glEnable(GL_MAP1_VERTEX_3);  
}
```

02/21/2002

15-462 Graphics I

27

## Evaluating Coordinates

- Use a fixed number of points, `num_points`

```
void display()  
{  
    ...  
    glBegin(GL_LINE_STRIP);  
    for (i = 0; i <= num_points; i++)  
        glEvalCoord1f((GLfloat)i/(GLfloat)num_points);  
    glEnd();  
    ...  
}
```

02/21/2002

15-462 Graphics I

28

## Drawing the Control Points

- To illustrate Bezier curve

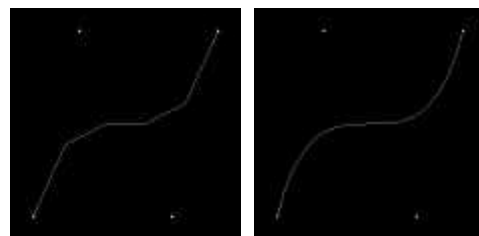
```
void display()  
{  
    ...  
    glPointSize(5.0);  
    glColor3f(1.0, 1.0, 0.0);  
    glBegin(GL_POINTS);  
    for (i = 0; i < 4; i++)  
        glVertex3fv(&ctrlpnts[i][0]);  
    glEnd();  
    glFlush();  
}
```

02/21/2002

15-462 Graphics I

29

## Resulting Images



n = 5

n = 20

02/21/2002

15-462 Graphics I

30

## Bezier Surfaces

- Create evaluator in two parameters u and v

```
glMap2f(GL_MAP2_VERTEX_3,
        u0, u1, ustride, uorder,
        v0, v1, vstride, vorder, point_array);
```

- Enable, also automatic calculation of normal

```
glEnable(GL_MAP2_VERTEX_3);
glEnable(GL_AUTO_NORMAL);
```

- Evaluate at parameters u and v

```
glEvalCoord2f(u, v);
```

02/21/2002

15-462 Graphics I

31

## Grids

- Convenience for uniform evaluators
- Define grid (nu = number of u division)

```
glMapGrid2f(nu, u0, u1, nv, v0, v1);
```

- Evaluate grid

```
glEvalMesh2(mode, i0, i1, k0, k1);
```

- *mode* = GL\_POINT, GL\_LINE, or GL\_FILL
- *i* and *k* define subrange

02/21/2002

15-462 Graphics I

32

## Example: Bezier Surface Patch

- Use 16 control points

```
GLfloat ctrlpnts[4][4][3] = {...};
```

- Initialize 2-dimensional evaluator

```
void init(void)
{
    ...
    glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
            0, 1, 12, 4, &ctrlpnts[0][0][0]);
    glEnable(GL_MAP2_VERTEX_3);
    glEnable(GL_AUTO_NORMAL);
    glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);
}
}
```

02/21/2002

15-462 Graphics I

33

## Evaluating the Grid

- Use full range

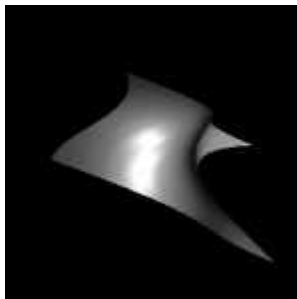
```
void display(void)
{
    ...
    glPushMatrix();
    glRotatef(85.0, 1.0, 1.0, 1.0);
    glEvalMesh2(GL_FILL, 0, 20, 0, 20);
    glPopMatrix();
    glFlush();
}
}
```

02/21/2002

15-462 Graphics I

34

## Resulting Image



02/21/2002

15-462 Graphics I

35

## NURBS Functions

- Higher-level interface
  - Implemented in GLU using evaluators
  - Create a NURBS renderer
- ```
theNurb = gluNewNurbsRenderer();
```
- Set NURBS properties

```
gluNurbsProperty(theNurb, GLU_DISPLAY_MODE, GLU_FILL);
gluNurbsCallback(theNurb, GLU_ERROR, nurbsError);
```

02/21/2002

15-462 Graphics I

36

## Displaying NURBS Surfaces

- Specify knot arrays for splines

```
GLfloat knots[8] = {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0};
gluBeginSurface(theNurb);
gluNurbsSurface(theNurb,
               8, knots, 8, knots,
               4 * 3, 3, &ctrlpoints[0][0][0],
               4, 4, GL_MAP2_VERTEX_3);
gluEndSurface(theNurb);
```

- For more see [Red Book, Ch. 12]

02/21/2002

15-462 Graphics I

37

## Summary

- Cubic B-Splines
- Nonuniform Rational B-Splines (NURBS)
- Rendering by Subdivision
- Curves and Surfaces in OpenGL

02/21/2002

15-462 Graphics I

38

## Reminders

- Assignment 3 due tonight
- Assignment 4 out some time today
- Midterm will cover curves and surfaces
- Next Tuesday: Textures?
- Next Thursday: John Ketchpaw

02/21/2002

15-462 Graphics I

39