Bezier Curves and Surfaces (Redbook ch12)

27 March 2007
CMPT370
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review last time

- Bump mapping theory
- Creating a texture in OpenGL
  - Texture objects: glBindTexture()
  - Loading image data: glTexImage2D()
    - Using the framebuffer as a texture
- Applying a texture in OpenGL
  - Blending modes: glTexEnvf()
  - Texture coordinates: glTexCoord2f()
    - Auto-generated texcoords: glTexGen()
    - Spherical environmental mapping
  - Mip-maps

What's on for today

- Polynomial curves and surfaces
- Cubic polynomial curves:
  - Interpolating (4 points)
  - Hermite (2 points + 2 derivatives)
  - Bezier (2 interpolating end points + 2 midpoints)
  - Using Bezier evaluators in OpenGL

Parametric representation

- Recall a 1D curve in 3D can be represented as:

$$p(u)=\begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix} \qquad p'(u)=\begin{bmatrix} x'(u) \\ y'(u) \\ z'(u) \end{bmatrix}$$

- p'(u) is the tangent (velocity) vector
  - Usually limit u to interval [0,1] for simplicity
- For surfaces we have two parameters (u, v):

$$p(u,v)=\begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix} \qquad \frac{\partial p}{\partial u}(u,v)=\begin{bmatrix} \partial x/\partial u \\ \partial y/\partial u \\ \partial z/\partial u \end{bmatrix} \qquad \frac{\partial p}{\partial v}(u,v)=\begin{bmatrix} \partial x/\partial v \\ \partial y/\partial v \\ \partial z/\partial v \end{bmatrix}$$

Polynomial curves

- Restrict the functions $x(u)$, $y(u)$, $z(u)$ to be polynomial (of degree n) in u:

$$\boldsymbol{p}(u) = \sum_{k=0}^{n} \boldsymbol{c}_k u^k$$

  - Each coefficient c_k is a 3-vector
  - u^k are the $n+1$ basis functions
  - Often choose $n=3$: cubic polynomial
    - k=0..3, (x,y,z): need 12 numbers
- Similarly for surfaces:

$$\boldsymbol{p}(u,v) = \sum_{j=0}^{n} \sum_{k=0}^{n} \boldsymbol{c}_{jk} u^j v^k$$

Interpolating Cubic Polynomials

- Simplest case, but rarely used in practice
- Four control points $p_0, \ldots, p_3$
- Fit a cubic polynomial through them
  - Space u evenly: $p_0 = p(0)$, $p_1 = p(1/3)$, ...

$$p_0 = p(0) = c_0$$

$$p_1 = p\left(\frac{1}{3}\right) = c_0 + \left(\frac{1}{3}\right) c_1 + \left(\frac{1}{3}\right)^2 c_2 + \left(\frac{1}{3}\right)^3 c_3$$

$$p_2 = p\left(\frac{2}{3}\right) = c_0 + \left(\frac{2}{3}\right) c_1 + \left(\frac{2}{3}\right)^2 c_2 + \left(\frac{2}{3}\right)^3 c_3$$

$$p_3 = p(1) = c_0 + c_1 + c_2 + c_3$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \left(\frac{1}{3}\right) & \left(\frac{1}{3}\right)^2 & \left(\frac{1}{3}\right)^3 \\ 1 & \left(\frac{2}{3}\right) & \left(\frac{2}{3}\right)^2 & \left(\frac{2}{3}\right)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Geometry matrix

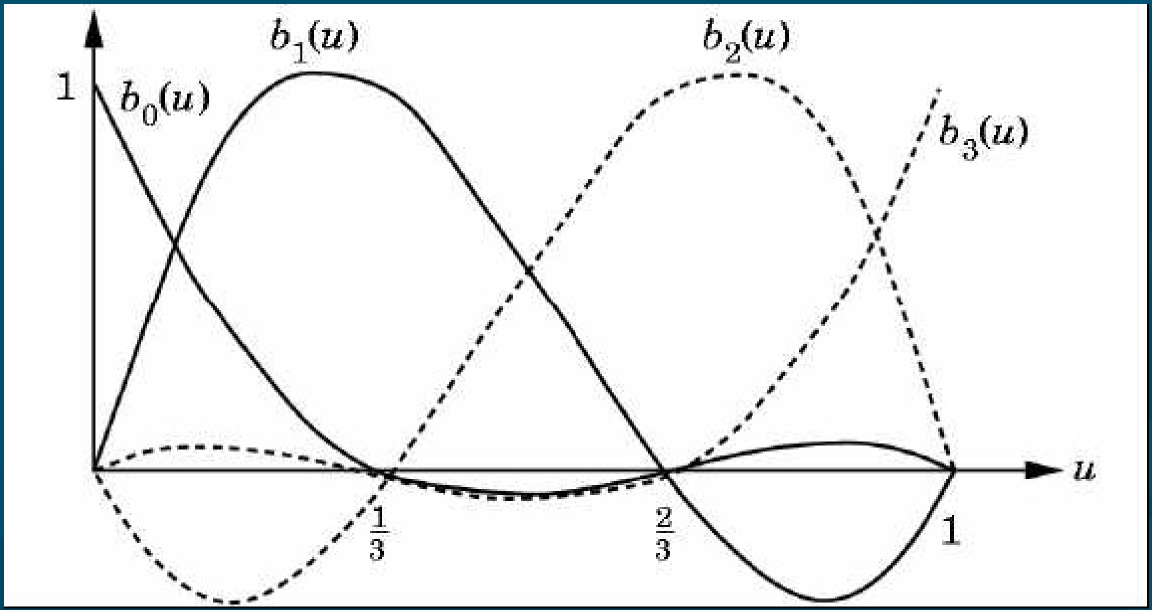
- Invert this matrix to get the geometry matrix
  - Multiply the geometry matrix by the four control points to get the coefficients

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -5.5 & 9 & -4.5 & 1 \\ 9 & -22.5 & 18 & -4.5 \\ -4.5 & 13.5 & -13.5 & 4.5 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

- The coefficients define the cubic polynomial that interpolates these control points
  - Can render, e.g., by using many small line segments (GL_LINE_STRIP)

Blending functions

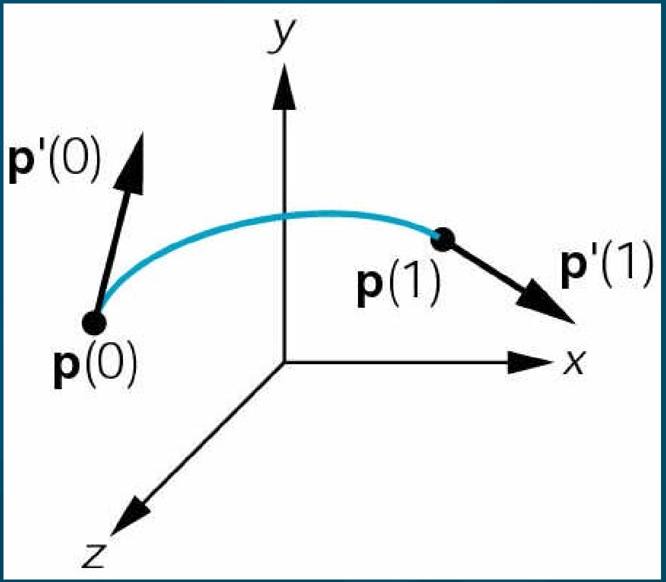
- We can also look at the contribution each control point makes to the final curve
- For interpolating cubics:
  - $p(u) = b_0(u)\ p_0 + b_1(u)\ p_1 + b_2(u)\ p_2 + b_3(u)\ p_3$



Hermite polynomial curves

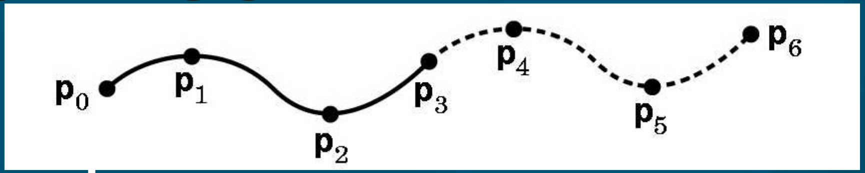
- Another way of defining cubic polynomials
- Specify start+end position+velocity
  - Also 12 numbers
- In matrix form:

$$\begin{bmatrix} p_0 \\ p_0' \\ p_3 \\ p_3' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

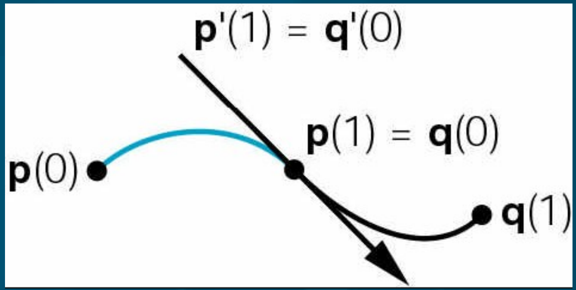


  - Invert to get Hermite geometry matrix from which we get the coefficients

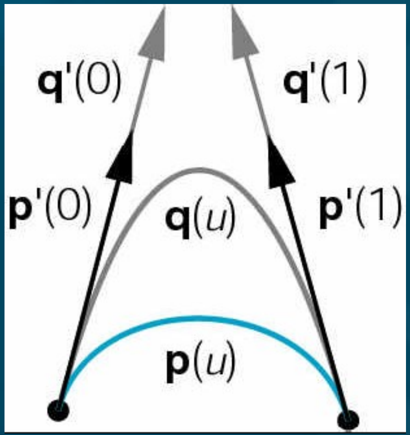
Joining polynomial curves



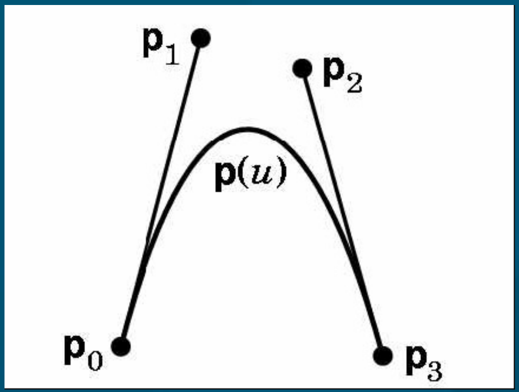
- Each segment has 4 control points
  - $\{p_0, p_1, p_2, p_3\}, \{p_3, p_4, p_5, p_6\}, \ldots$
- Kinds of continuity: differential:
  - C^0: touching but may have corner
  - C^1: derivatives match (Hermite)
  - C^2: curvatures match



- Geometric continuity:
  - G^1: velocity vectors in same direction but not necessarily same magnitude



Bezier curves

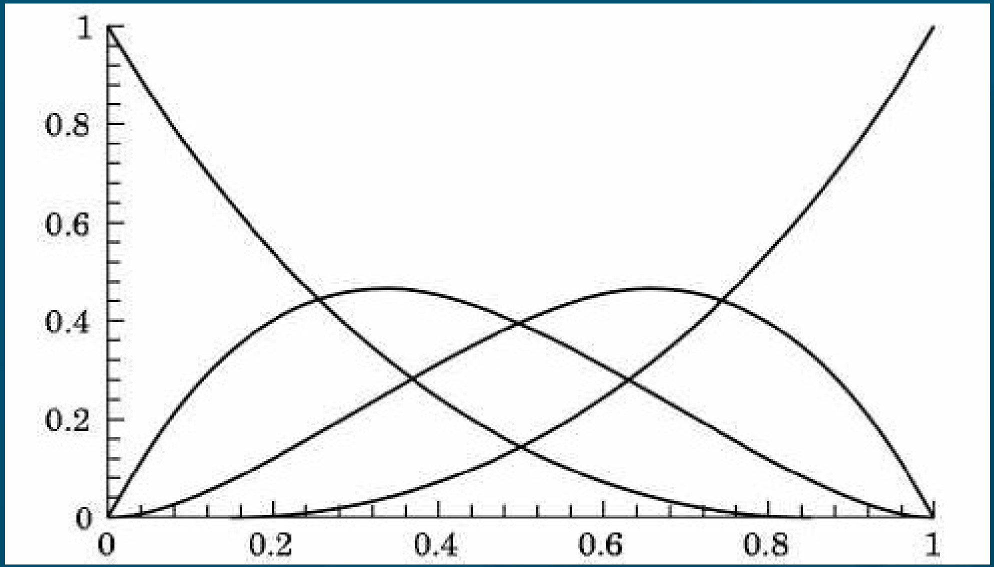


- Widely used, provided in OpenGL
- Use control points to indicate tangent vectors
  - Does not interpolate middle control points!
  - $p'(0) = 3(p_1 - p_0)$, $p'(1) = 3(p_3 - p_2)$
- p_0, p_3 specify start+end position
- start+end velocity derived from control points
- Use Hermite form
- C^0 but not C^1

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

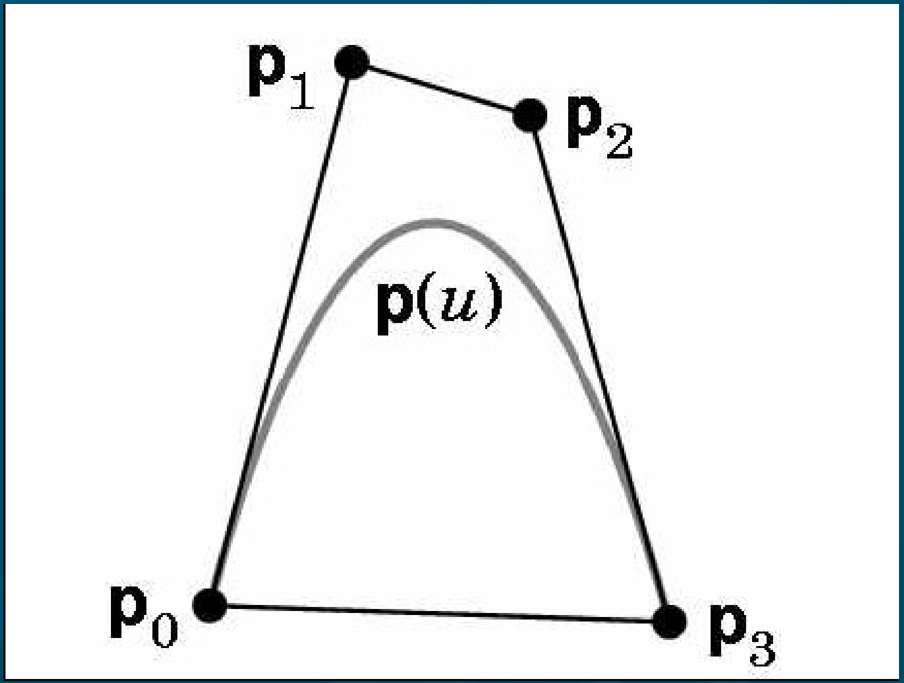
Bezier blending functions

- Blending functions are smooth polynomials
  - $b_0(u) = (1-u)^3$
  - $b_1(u) = 3u(1-u)^2$
  - $b_2(u) = 3u^2(1-u)$
  - $b_3(u) = 3u^3$



Convex hull property

- Why the factor of 3 in the definition of Bezier curves?
  - $p'(0) = 3(p_1 - p_0)$
  - $p'(1) = 3(p_3 - p_2)$
- Ensures that the curve is contained within the convex hull of the four control points



Bezier evaluators in OpenGL

- Specify array (1D or 2D) of control points:
  - GLfloat ctrlpoints[4][3] = { {-4.0, -4.0, 0.0}, ...
- Create a Bezier evaluator: *(type=GL_MAP1_VERTEX_3)*
  - glMap1f(*type*, u_{min}, u_{max}, *stride*, *order*, *points*);
- Enable the evaluator:
  - glEnable(*type*);
- Evaluate the Bezier at a particular u/v:
  - glEvalCoord1f((GLfloat) u);
    - Use this instead of glVertex(), e.g., within glBegin(GL_LINE_STRIP)