

Parametric Curves

**CSCI 4229/5229
Computer Graphics
Fall 2013**

Parametric Curves

- $x(t) = p_x(t), y(t) = p_y(t), z(t) = p_z(t), w(t) = p_w(t)$
- Often $p(t)$ is a polynomial
- Generally t in $[0,1]$
- Avoids problems such as lines parallel to axes
- Works for any number of dimensions
- Can be used to generate vertexes, colors, texture coordinates, normals, etc.

Bernstein Polynomials

- Bernstein Polynomials

- $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$

- Sums to one $\sum_{i=0}^n B_i^n(t) = 1$

- Cubic Bernstein Polynomials

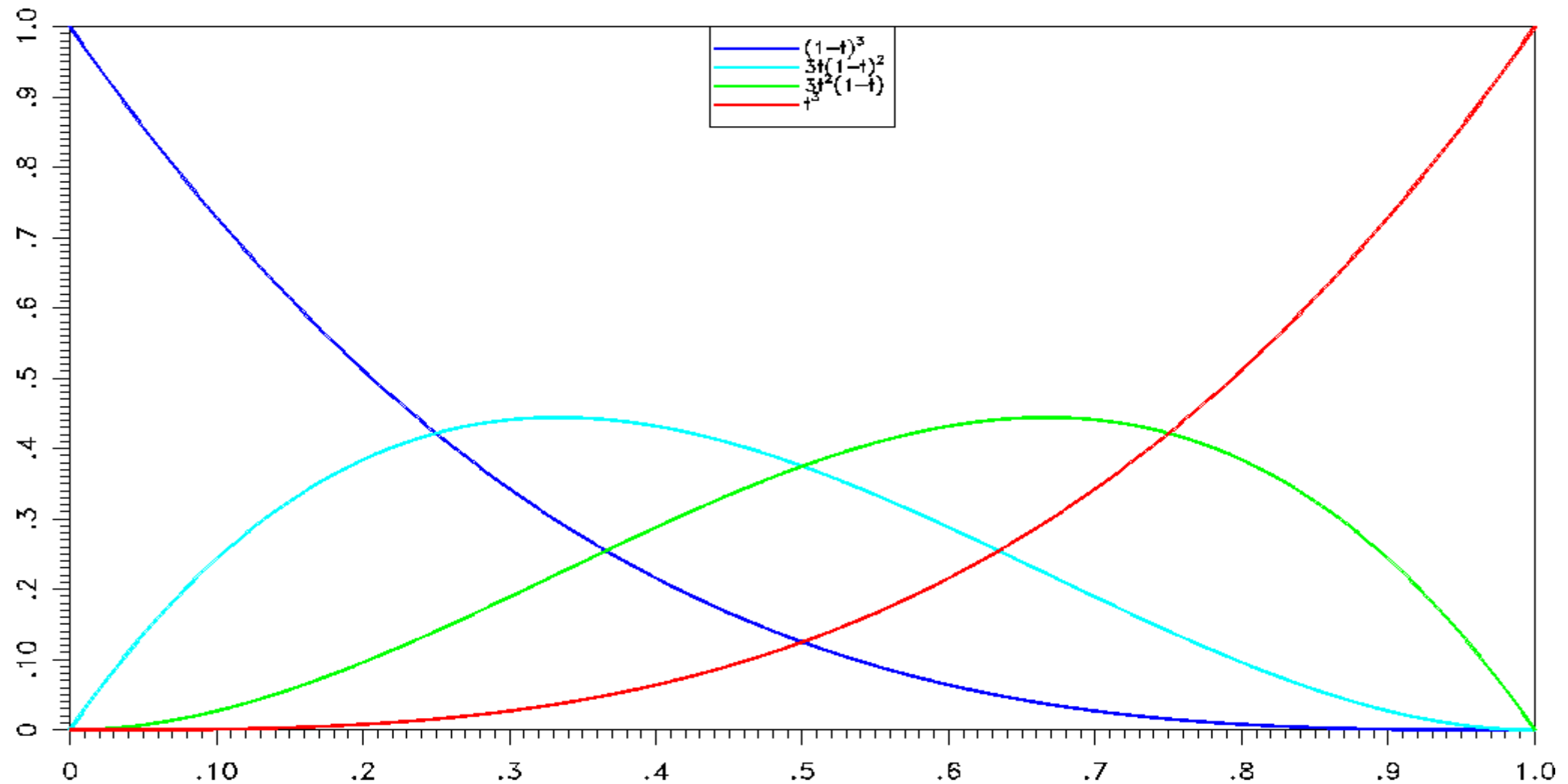
- $B_0^3(t) = (1-t)^3$

- $B_1^3(t) = 3t(1-t)^2$

- $B_2^3(t) = 3t^2(1-t)$

- $B_3^3(t) = t^3$

Cubic Bernstein Polynomials



Bézier Curves

- $C_n(t) = \sum_{i=0}^n B_i^n(t) P_i, \quad t \in [0,1]$
- P_i are points in 2D, 3D or 4D
- Convex linear combination of points P_i
 - Entire curve is in convex hull of points ($\sum_{i=0}^n B_i^n(t)=1$)
 - $B_0^n(0) = 1$, so starts at P_0
 - $B_n^n(1) = 1$, so ends at P_n
 - Tangential to P_0-P_1 and P_n-P_{n-1}
- Curve is smooth and differentiable

Curves in OpenGL

- One-dimensional Evaluators
- Can be used to generate vertexes, normals, colors and textures
- Curve defined analytically using Bezier curves
- Evaluated at discrete points and rendered using straight line segments

Curves in OpenGL

- `glEnable()`
 - Enables types of data to generate
- `glMap1d()`
 - Defines control points and domain
- `glEvalCoord1d()`
 - Generates a data point
- `glMapGrid1d()` & `glEvalMesh1()`
 - Generates a series of data points
- Deprecated in OpenGL (do this manually)

glMap1d(type,Umin,Umax,stride,order,points)

- *type* of data to generate
 - GL_MAP1_VERTEX_3[4]
 - GL_MAP1_NORMAL
 - GL_MAP1_COLOR_4
 - GL_MAP1_TEXTURE_COORD_[1-4]
- *Umin&Umax* are limits of parameter (often 0&1)
- *stride* is the number of coordinates in data (3 or 4)
- *order* is the order of the curve (4=cubic)
- *points* is the array of data points
- **Remember to also call glEnable()**

glEvalCoord1d(u)

- Generate one vertex for each glMap1d() type currently active (e.g. texture, normal, vertex)
- To generate the whole curve, call glEvalCoord1d() once for each vertex
- Exercise entire parameter space
 - u from Umin to Umax (0 to 1)

Generating a complete curve

- `glMapGrid1d(N , U1 , U2)`
- `glEvalMesh1(mode , N1 , N2)`
- This is equivalent to

```
glBegin(mode);
for (i=N1;i<=N2;i++)
    glEvalCoord1(U1 + i*(U2-U1)/N);
glEnd();
```

Interpolation with Bézier Curves

- We have 4 points we want the curve to pass through P_0, P_1, P_2 & P_3
- What should control points R_0, R_1, R_2 & R_3 be?

$$P_0 = R_0 B_0^3(0) + R_1 B_1^3(0) + R_2 B_2^3(0) + R_3 B_3^3(0)$$

$$P_1 = R_0 B_0^3\left(\frac{1}{3}\right) + R_1 B_1^3\left(\frac{1}{3}\right) + R_2 B_2^3\left(\frac{1}{3}\right) + R_3 B_3^3\left(\frac{1}{3}\right)$$

$$P_2 = R_0 B_0^3\left(\frac{2}{3}\right) + R_1 B_1^3\left(\frac{2}{3}\right) + R_2 B_2^3\left(\frac{2}{3}\right) + R_3 B_3^3\left(\frac{2}{3}\right)$$

$$P_3 = R_0 B_0^3(1) + R_1 B_1^3(1) + R_2 B_2^3(1) + R_3 B_3^3(1)$$

Relationship between P and R

$$\begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix} = \begin{pmatrix} B_0^3(0) & B_1^3(0) & B_2^3(0) & B_3^3(0) \\ B_0^3(\frac{1}{3}) & B_1^3(\frac{1}{3}) & B_2^3(\frac{1}{3}) & B_3^3(\frac{1}{3}) \\ B_0^3(\frac{2}{3}) & B_1^3(\frac{2}{3}) & B_2^3(\frac{2}{3}) & B_3^3(\frac{2}{3}) \\ B_0^3(1) & B_1^3(1) & B_2^3(1) & B_3^3(1) \end{pmatrix} \begin{pmatrix} R_0 \\ R_1 \\ R_2 \\ R_3 \end{pmatrix}$$

$$\begin{pmatrix} R_0 \\ R_1 \\ R_2 \\ R_3 \end{pmatrix} = \begin{pmatrix} B_0^3(0) & B_1^3(0) & B_2^3(0) & B_3^3(0) \\ B_0^3(\frac{1}{3}) & B_1^3(\frac{1}{3}) & B_2^3(\frac{1}{3}) & B_3^3(\frac{1}{3}) \\ B_0^3(\frac{2}{3}) & B_1^3(\frac{2}{3}) & B_2^3(\frac{2}{3}) & B_3^3(\frac{2}{3}) \\ B_0^3(1) & B_1^3(1) & B_2^3(1) & B_3^3(1) \end{pmatrix}^{-1} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

Bézier Interpolation Matrix

- Selected so $t=1/3$ and $t=2/3$ maps to P_1 & P_2
- Local function (depends only on P_0, P_1, P_2, P_3)

$$\begin{array}{l}
 B_0^3(t) = (1-t)^3 \\
 B_1^3(t) = 3t(1-t)^2 \\
 B_2^3(t) = 3t^2(1-t) \\
 B_3^3(t) = t^3
 \end{array}
 \quad
 \begin{array}{l}
 B_0^3(0) = 1 \\
 B_1^3(0) = 0 \\
 B_2^3(0) = 0 \\
 B_3^3(0) = 0
 \end{array}
 \quad
 \begin{array}{l}
 B_0^3(\frac{1}{3}) = \frac{8}{27} \\
 B_1^3(\frac{1}{3}) = \frac{4}{9} \\
 B_2^3(\frac{1}{3}) = \frac{2}{9} \\
 B_3^3(\frac{1}{3}) = \frac{1}{27}
 \end{array}
 \quad
 \begin{array}{l}
 B_0^3(\frac{2}{3}) = \frac{1}{27} \\
 B_1^3(\frac{2}{3}) = \frac{2}{9} \\
 B_2^3(\frac{2}{3}) = \frac{4}{9} \\
 B_3^3(\frac{2}{3}) = \frac{8}{27}
 \end{array}
 \quad
 \begin{array}{l}
 B_0^3(1) = 0 \\
 B_1^3(1) = 0 \\
 B_2^3(1) = 0 \\
 B_3^3(1) = 1
 \end{array}$$

$$\begin{pmatrix} 1 & \frac{8}{27} & \frac{1}{27} & 0 \\ 0 & \frac{4}{9} & \frac{2}{9} & 0 \\ 0 & \frac{2}{9} & \frac{4}{9} & 0 \\ 0 & \frac{1}{27} & \frac{8}{27} & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{5}{6} & 3 & -\frac{3}{2} & \frac{1}{3} \\ \frac{1}{3} & -\frac{3}{2} & 3 & -\frac{5}{6} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Extending to More Points

- Two curves P_0, P_1, P_2, P_3 and P_4, P_5, P_6, P_7
- Bézier curves pass through $P_0 \& P_3$ and $P_4 \& P_7$, so the curve will be continuous if $P_3 = P_4$
- Bézier curves are tangential to $P_1 - P_0$ & $P_2 - P_3$ and $P_5 - P_4$ & $P_6 - P_7$, so the curve will be smooth if $P_3 = P_4$ and $P_2 - P_3$ and $P_5 - P_4$, therefore $P_5 = 2P_3 - P_2$

Splines

- Traditionally a long, thin, flexible piece of wood or metal used to describe a smooth curve
 - Used in building boats, airplanes, etc.
- Held down by *ducks* or *whales*
- Mathematical equivalents
 - Natural Cubic Spline
 - Weights called *knots*
 - Piecewise polynomial



Parametric Splines

- Three or four splines, one for each component
- Parameter t reach integer values at each knot
 - Cardinal spline
- Natural Cubic Spline
- Clamped Cubic Spline
- Quadratic Spline
- Hermite Spline

Cardinal Cubic Spline

$$S_j(t) = \frac{(j+1-t)^3 - (j+1-t)}{6}G_j + \frac{(t-j)^3 - (t-j)}{6}G_{j+1} + (j+1-t)F_j + (t-j)F_{j+1}$$

$$S'_j(t) = \frac{1 - 3(j+1-t)^2}{6}G_j + \frac{3(t-j)^2 - 1}{6}G_{j+1} + F_{j+1} - F_j$$

$$S''_j(t) = (j+1-t)G_j + (t-j)G_{j+1}$$

$$S_j(j) = F_j$$

$$S_{j-1}(j) = F_j$$

$$S''_j(j) = G_j$$

$$S''_{j-1}(j) = G_j$$

$$S'_j(j) = -\frac{1}{3}G_j - \frac{1}{6}G_{j+1} + F_{j+1} - F_j$$

$$S'_{j-1}(j) = \frac{1}{6}G_{j-1} - \frac{1}{3}G_j + F_j - F_{j-1}$$

$$S'_{j-1}(j) = S'_j(j)$$

$$\frac{1}{6}G_{j-1} + \frac{2}{3}G_j + \frac{1}{6}G_{j+1} = F_{j-1} - 2F_j + F_{j+1}$$

