Matrix Form for Cubic Bézier Curves

$$
\begin{aligned}
\mathbf{p}(u) = \; & (1-u)^3 \mathbf{p}_0 \\
& +3(1-u)^2 (u) \mathbf{p}_1 \\
& +3(1-u)(u)^2 \mathbf{p}_2 \\
& +(u)^3 \mathbf{p}_3
\end{aligned}
$$

$$
\mathbf{p}(u) = \begin{pmatrix} 1 & u & u^2 & u^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix}
$$

Bézier Tangents

Suppose we have a Bézier curve

$$
\mathbf{p}(u) = \sum_{i=0}^{n} \mathbf{p}_i B_i^n(u) \qquad 0 \le u \le 1
$$

The derivatives at the endpoints are

$$
\mathbf{p}'(0) = n(\mathbf{p}_1 - \mathbf{p}_0)
$$
$$
\mathbf{p}'(1) = n(\mathbf{p}_n - \mathbf{p}_{n-1})
$$

So in the cubic case we have:

$$
\mathbf{p}'(0) = 3(\mathbf{p}_1 - \mathbf{p}_0)
$$
$$
\mathbf{p}'(1) = 3(\mathbf{p}_3 - \mathbf{p}_2)
$$

Bézier–Hermite Conversion

This gives us a direct connection to Hermite splines

$$
\begin{aligned}
\overbrace{\mathbf{p}_0}^{\text{Hermite}} &= \overbrace{\mathbf{p}_0}^{\text{Bezier}} \\
\mathbf{p}_3 &= \mathbf{p}_3 \\
\mathbf{r}_0 &= 3(\mathbf{p}_1 - \mathbf{p}_0) \\
\mathbf{r}_3 &= 3(\mathbf{p}_3 - \mathbf{p}_2)
\end{aligned}
$$

Which we can write in matrix form:

$$
\begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_3 \\ \mathbf{r}_0 \\ \mathbf{r}_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix}
$$

Converting Between Cubic Spline Types

We saw a specific example of Bézier–Hermite conversion

$$
\begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_3 \\ \mathbf{r}_0 \\ \mathbf{r}_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix}
$$

Suppose we want to convert between two arbitrary splines

$$
\mathbf{u}^{\mathsf{T}} \mathbf{M}_1 \mathbf{G}_1 = \mathbf{u}^{\mathsf{T}} \mathbf{M}_2 \mathbf{G}_2
$$

Given geometry matrix $\mathbf{G}_1$ find equivalent $\mathbf{G}_2$ for other spline

$$
\mathbf{G}_2 = \mathbf{M}_2^{-1} \mathbf{M}_1 \mathbf{G}_1
$$

Classifying Continuity of Curves

Parametric Continuity — C^k

- each coordinate function is differentiable k times
- and they are continuous through k^{th} derivative

Geometric Continuity — G^k

- the curve itself is continuous up to order k
- independent of parameterization
- G^0 — two segments meet at same point
- G^1 — with same tangent
- G^2 — and same curvature

These two kinds of continuity are *not* always equivalent

Exercise: Bézier Continuity

Suppose that you're given two cubic Bézier control polygons

$$\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$$
$$\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$$

where the two curves $\mathbf{p}$ and $\mathbf{q}$ should be joined consecutively.

What constraints on these points are necessary to guarantee C^1 continuity between them?

Catmull–Rom Splines

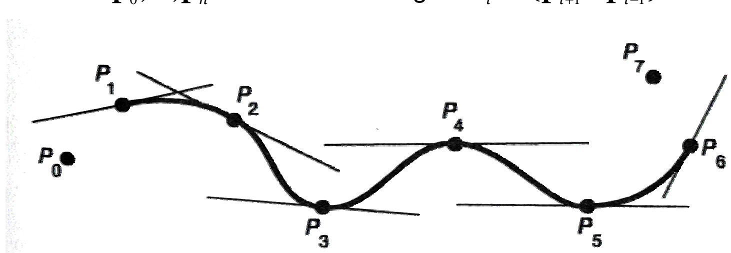
Given a set of points in space, suppose we want a spline that

- interpolates the data points [rules out Bézier]
- with C^1 continuity [Hermite: lots of tweaking]

This is a common situation in animation

We start with the given set of points

$\mathbf{p}_0, \ldots, \mathbf{p}_n$ define tangent $\mathbf{r}_i = s(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$



Catmull–Rom Splines

Typically, we pick $s = ½$ and we can derive a spline equation

$$\mathbf{p}(u) = \frac{1}{2}\begin{pmatrix} 1 & u & u^2 & u^3 \end{pmatrix} \begin{pmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{p}_{i-3} \\ \mathbf{p}_{i-2} \\ \mathbf{p}_{i-1} \\ \mathbf{p}_i \end{pmatrix}$$

More generally, we can use any tension parameter *s*

$$\mathbf{p}(u) = \begin{pmatrix} 1 & u & u^2 & u^3 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ -s & 0 & s & 0 \\ 2s & s-3 & 3-2s & -s \\ -s & 2-s & s-2 & s \end{pmatrix} \begin{pmatrix} \mathbf{p}_{i-3} \\ \mathbf{p}_{i-2} \\ \mathbf{p}_{i-1} \\ \mathbf{p}_i \end{pmatrix}$$

B-Splines

Like Catmull–Rom splines, start with sequence of points $\mathbf{p}_0,\ldots,\mathbf{p}_n$

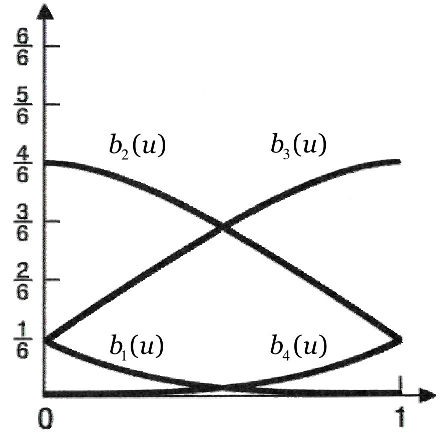
$$\mathbf{p}(u)=\frac{1}{6}\begin{pmatrix}1 & u & u^2 & u^3\end{pmatrix}\begin{bmatrix}1 & 4 & 1 & 0\\ -3 & 0 & 3 & 0\\ 3 & -6 & 3 & 0\\ -1 & 3 & -3 & 1\end{bmatrix}\begin{bmatrix}\mathbf{p}_{i-3}\\ \mathbf{p}_{i-2}\\ \mathbf{p}_{i-1}\\ \mathbf{p}_i\end{bmatrix}$$

Curves no longer interpolate control points
- points where segments actually meet are called knots
- for Hermite *et al* the knots were always control points

Lack of interpolation isn't a big problem for interactive design
- but it's hard to predict curve just based on points coordinates

B-Spline Basis Functions

$$b_1(u)\mathbf{p}_{i-3}+b_2(u)\mathbf{p}_{i-2}+b_3(u)\mathbf{p}_{i-1}+b_4(u)\mathbf{p}_i$$



Non-negative functions
- implies convex hull property

$$b_1(u)=\frac{1}{6}(1-u)^3$$
$$b_2(u)=\frac{1}{6}\left(3u^3-6u^2+4\right)$$
$$b_3(u)=\frac{1}{6}\left(-3u^3+3u^2+3u+1\right)$$
$$b_4(u)=\frac{1}{6}u^3$$

Drawing Spline Curves

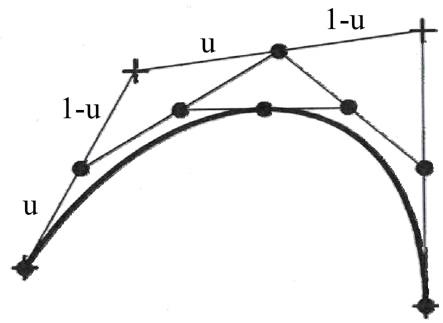
Method #1 — Direct evaluation
- we have a function that generates points on the curve
- vary parameter u between 0 and 1
- substitute into formula and compute a position
- connect consecutive points with line segments

Method #1a — Direct evaluation with forward differencing
- instead of evaluating polynomials directly
- incrementalize polynomial to cut down on multiplies

This approach has some problems
- uniform parameter spacing is not uniform in space
- length of segments will vary over line
- control over length is important
  - too long makes jagged curves; too short is too slow to draw

Bézier Curve Subdivision

Subdividing control polyline
- produces two new control polylines for each half of the curve
- defines the same curve
- all control points are closer to the curve
- this is handy for drawing



Drawing Spline Curves

Method #2 — Recursive subdivision

- starting with initial control polyline, recursively subdivide
- each subdivision produces points closer to curve
- keep doing this until the segments are good enough
  - until they're short enough (roughly constant line size)
  - or curve is locally flat enough (fewer lines in straight regions)

And we only have to write this code once!

- we've formulated a uniform representation for splines
- all we need to know is the basis & geometry matrices

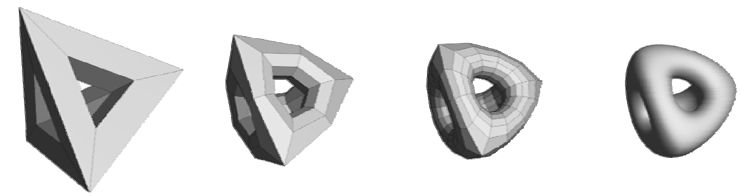
Modeling By Subdivision

Recall that we can draw spline curves via subdivision

- start with the control polyline
- recursively subdivide until "smooth enough"
- and draw the individual line segments

We can actually use this as a modeling primitive

- define the curve as limit of infinite number of subdivision steps
- throw out all our polynomials



Developing Subdivision Curves

Assume that we have some control polygon

- a closed piecewise-linear curve in the plane

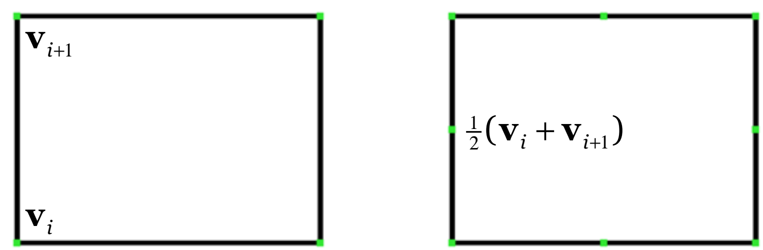
Need two fundamental operations:

- Linear Subdivision — introduce new vertices
- Linear Smoothing — modify positions of vertices

Linear Subdivision of Curves

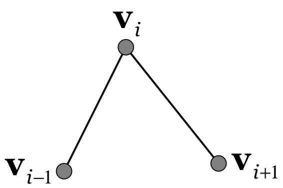
Split each edge of the curve at its barycenter (midpoint)

- thus doubling the number of vertices



Linear Smoothing of Curves

Reposition each vertex at weighted combination of neighbors



$$\mathbf{v}_i' = \alpha_1 \mathbf{v}_{i-1} + \alpha_2 \mathbf{v}_i + \alpha \mathbf{v}_{i+1}$$

$$\sum_i \alpha_i = 1$$

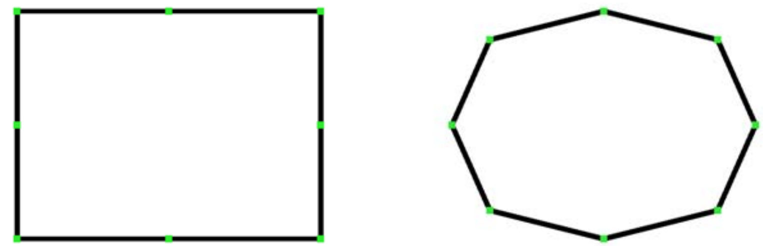
Can also rewrite the above in a matrix form

$$\mathbf{v}_i' = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix} \begin{bmatrix} \mathbf{v}_{i-1} \\ \mathbf{v}_i \\ \mathbf{v}_{i+1} \end{bmatrix}$$

Linear Smoothing of Curves

We are generally interested in symmetric weighting schemes

$$\mathbf{v}_i' = \left(\frac{1-\alpha}{2}\right) \mathbf{v}_{i-1} + \alpha \mathbf{v}_i + \left(\frac{1-\alpha}{2}\right) \mathbf{v}_{i+1}$$



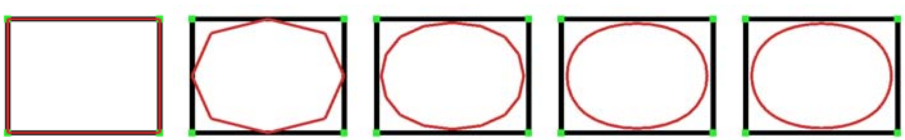
weights $\left[\frac{1}{4} \quad \frac{1}{2} \quad \frac{1}{4}\right]$

Creating Smooth Curves by Subdivision

Alternately repeat subdivision & smoothing operators

- converges to some limit curve (determined by weights)

For weights [¼ ½ ¼] resulting curve is piecewise B-spline!



0 1 2 3 4

Subdivision Level