University of British Columbia

CPSC 414 Computer Graphics

Curves

Week 13, Mon 24 Nov 2003

News

- final
  - LSK 200, noon Tue Dec 9
  - must have photo ID (student ID best)

- hw1, proj2 grades out

- TA lab hours as usual this week
- reminder: my office hours in lab today
  - 10:30-11:30

Schedule: Lab Hours for P3

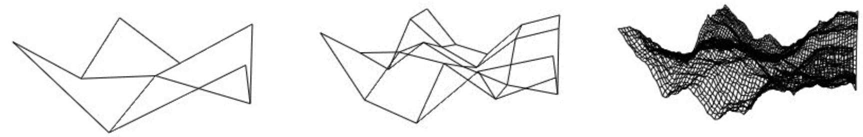
- Mon Dec 1
  – AG 10-12, AW 12-2
- Tue Dec 2
  – AG 10-12, AW 12-2, TM 2-4
- Wed Dec 3
  – AW 1-2, PZ 2-4
- Thu Dec 4
  – AG 11-1
- Fri Dec 5
  – AG 10-11, PZ 11-1

Schedule: Lectures

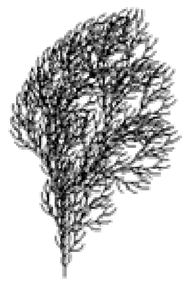
- Mon (today)
  - curves
- Wed
  - advanced rendering, final review
- Fri
  - evaluations, 3D CG in movies
    - Pixar shorts, The Shape of Space

Procedural Approaches recap

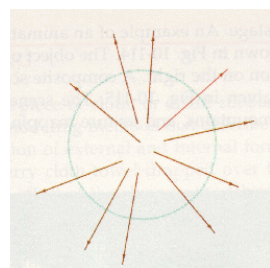
- fractal landscapes



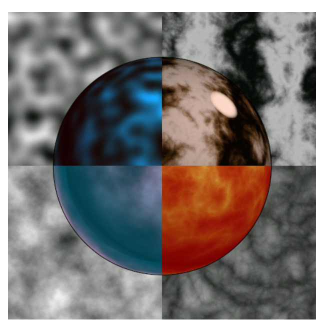
- L-systems



- particle systems



- Perlin noise



University of British Columbia

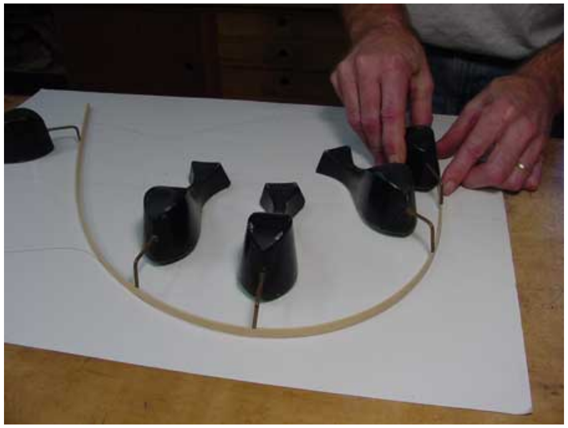
CPSC 414 Computer Graphics

Curves recap

Splines

- *spline* is parametric curve defined by *control points*
  - *knots:* control points that lie on curve
  - engineering drawing: spline was flexible wood, control points were physical weights

A Duck (weight)



Ducks trace out curve

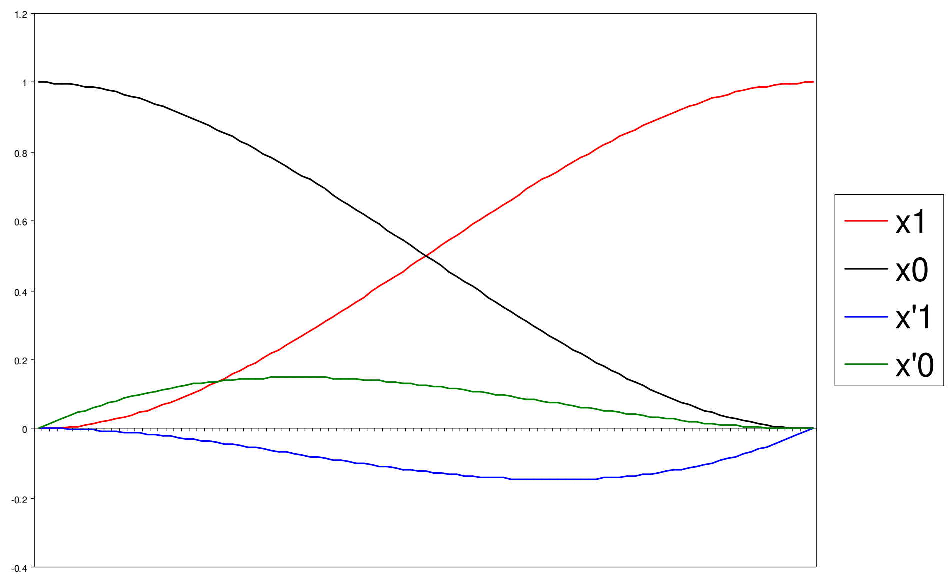
Hermite Spline

- user provides
  - endpoints
  - derivatives at endpoints

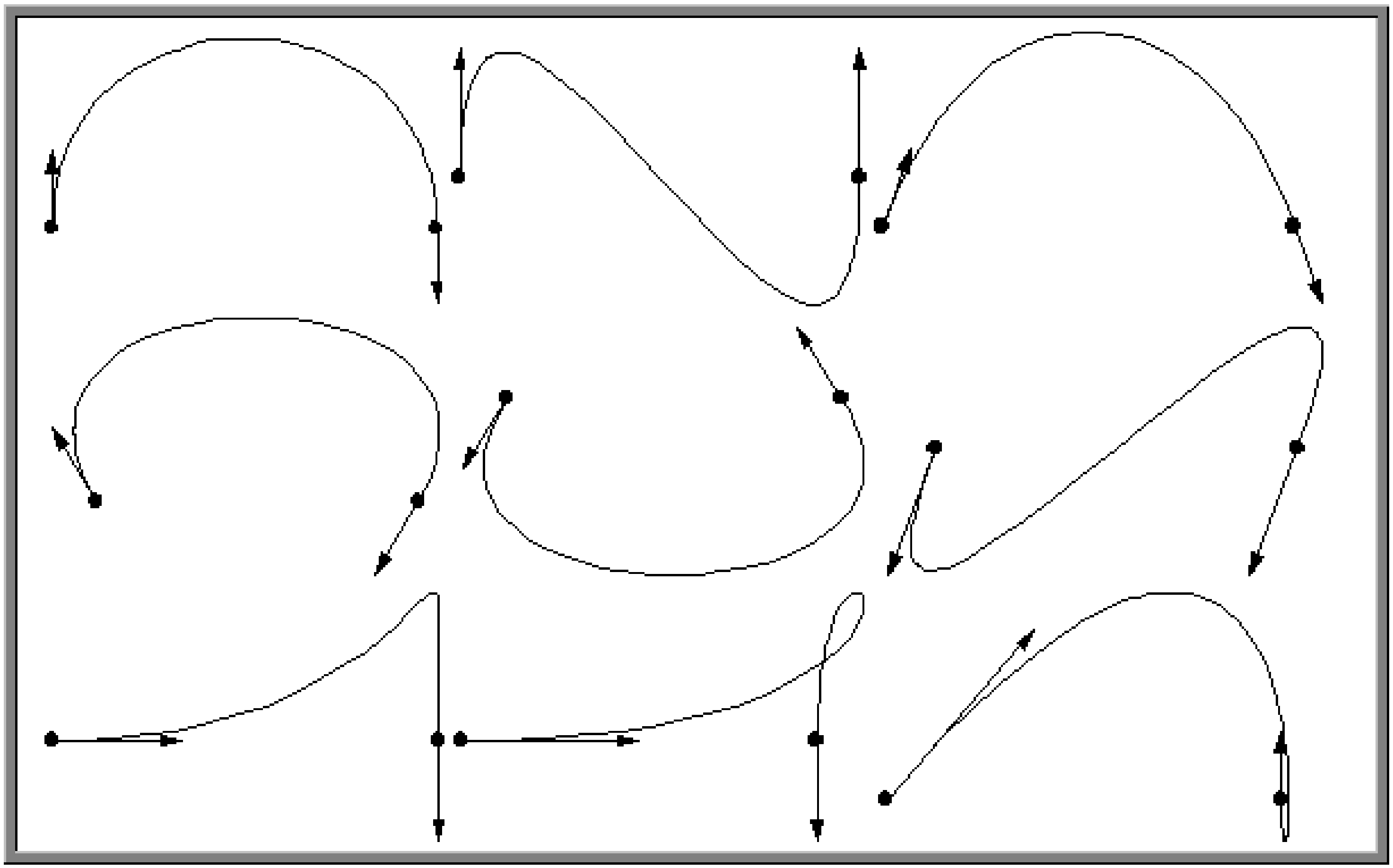
$$x = \begin{bmatrix} x_1 & x_0 & x_1' & x_0' \end{bmatrix} \begin{bmatrix} -2 & 3 & 0 & 0 \\ 2 & -3 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

Basis Functions

- a point on a Hermite curve is obtained by multiplying each control point by some function and summing
- functions are called *basis functions*



Sample Hermite Curves



University of British Columbia

CPSC 414 Computer Graphics

Curves

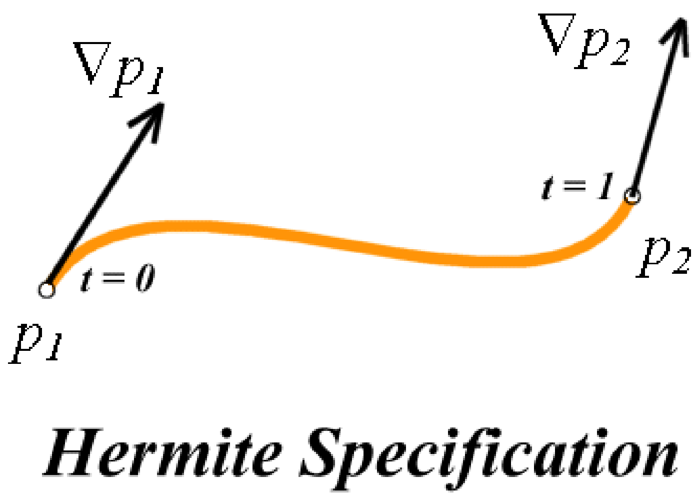
Splines in 2D and 3D

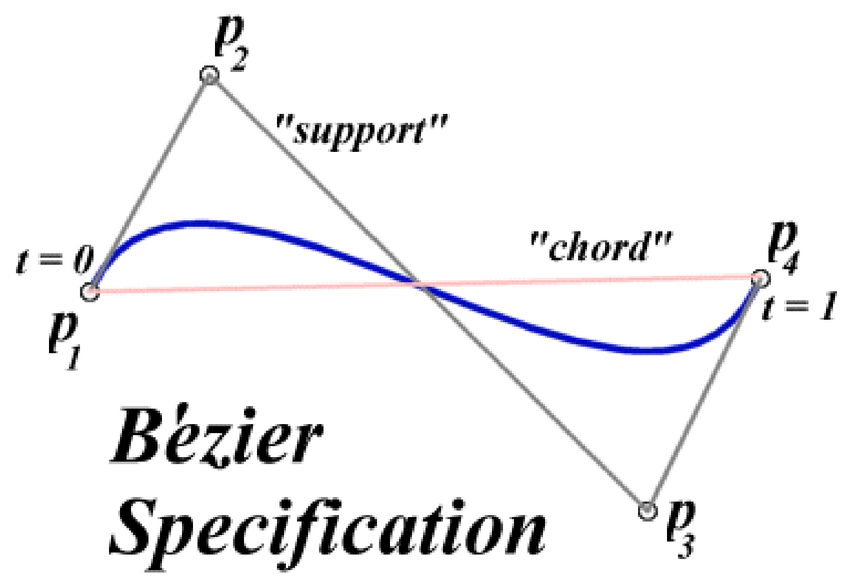
- so far, defined only 1D splines: $x=f(t:x_0,x_1,x'_0,x'_1)$
- for higher dimensions, define control points in higher dimensions (that is, as vectors)

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_1 & x_0 & x'_1 & x'_0 \\ y_1 & y_0 & y'_1 & y'_0 \\ z_1 & z_0 & z'_1 & z'_0 \end{bmatrix} \begin{bmatrix} -2 & 3 & 0 & 0 \\ 2 & -3 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

Bézier Curves

- similar to Hermite, but more intuitive definition of endpoint derivatives
- four control points, two of which are knots





Bézier Curves

- derivative values of Bezier curve at knots dependent on adjacent points

$$\nabla p_1 = 3(p_2 - p_1)$$
$$\nabla p_4 = 3(p_4 - p_3)$$

Bézier vs. Hermite

- can write Bezier in terms of Hermite
  - note: just matrix form of previous equations

$$\underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

Bézier vs. Hermite

- Now substitute this in for previous Hermite

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

Bézier Basis, Geometry Matrices

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Bezier}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

- but why is $\mathrm{M}_{\mathrm{Bezier}}$ a good basis matrix?

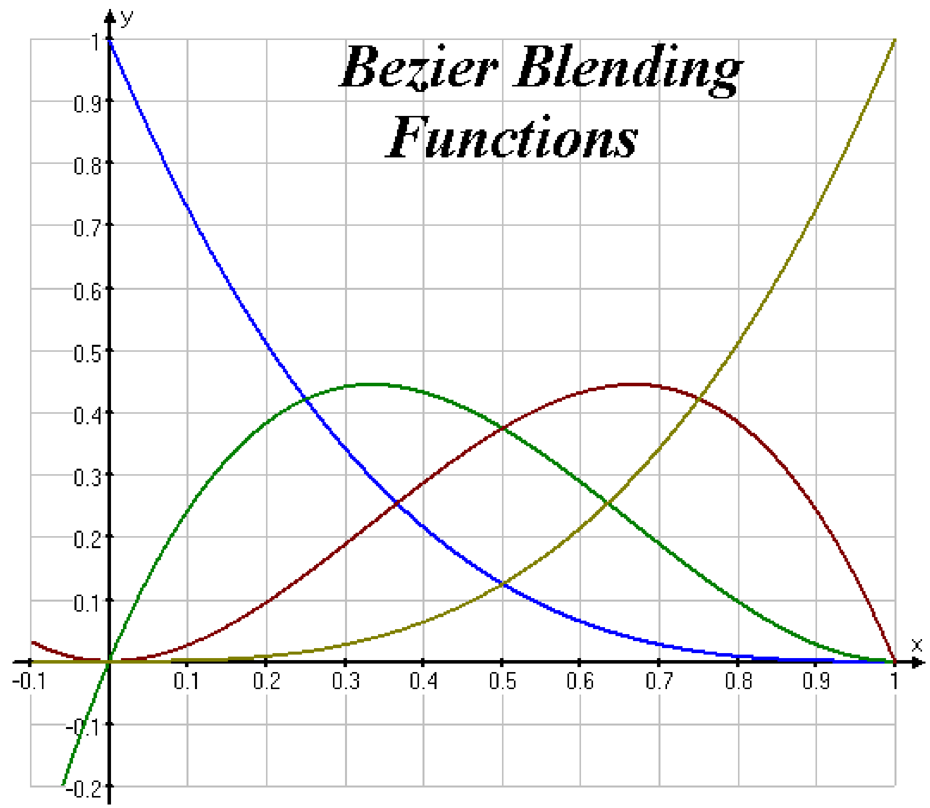
Bézier Blending Functions

- look at blending functions
- family of polynomials called order-3 Bernstein polynomials
  - C(3, k) t^k $(1\text{-}t)^{3\text{-}k}$; 0<= k <= 3
  - all positive in interval [0,1]
  - sum is equal to 1

$$p(t) = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}^{\mathbf{T}} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

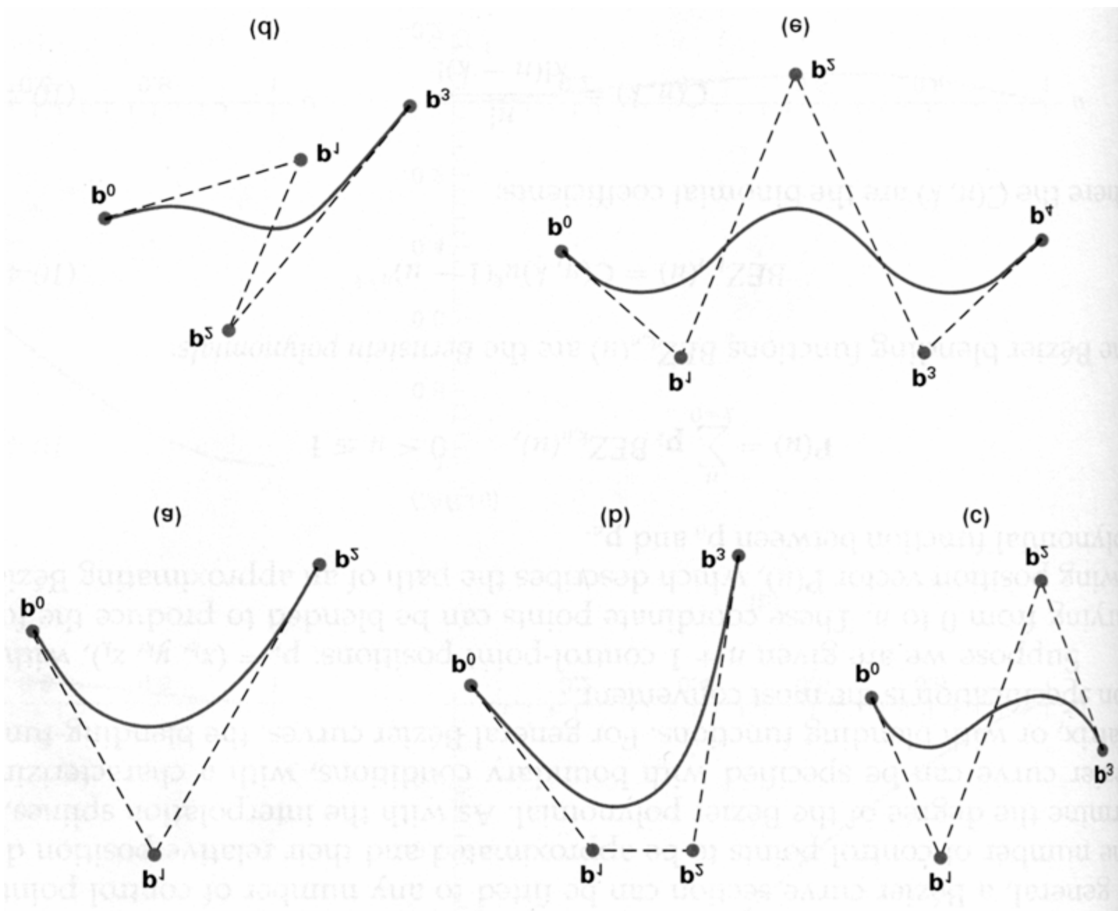
Bézier Blending Functions

- every point on curve is linear combination of control points
- weights of combination are all positive
- sum of weights is 1
- therefore, curve is a convex combination of the control points



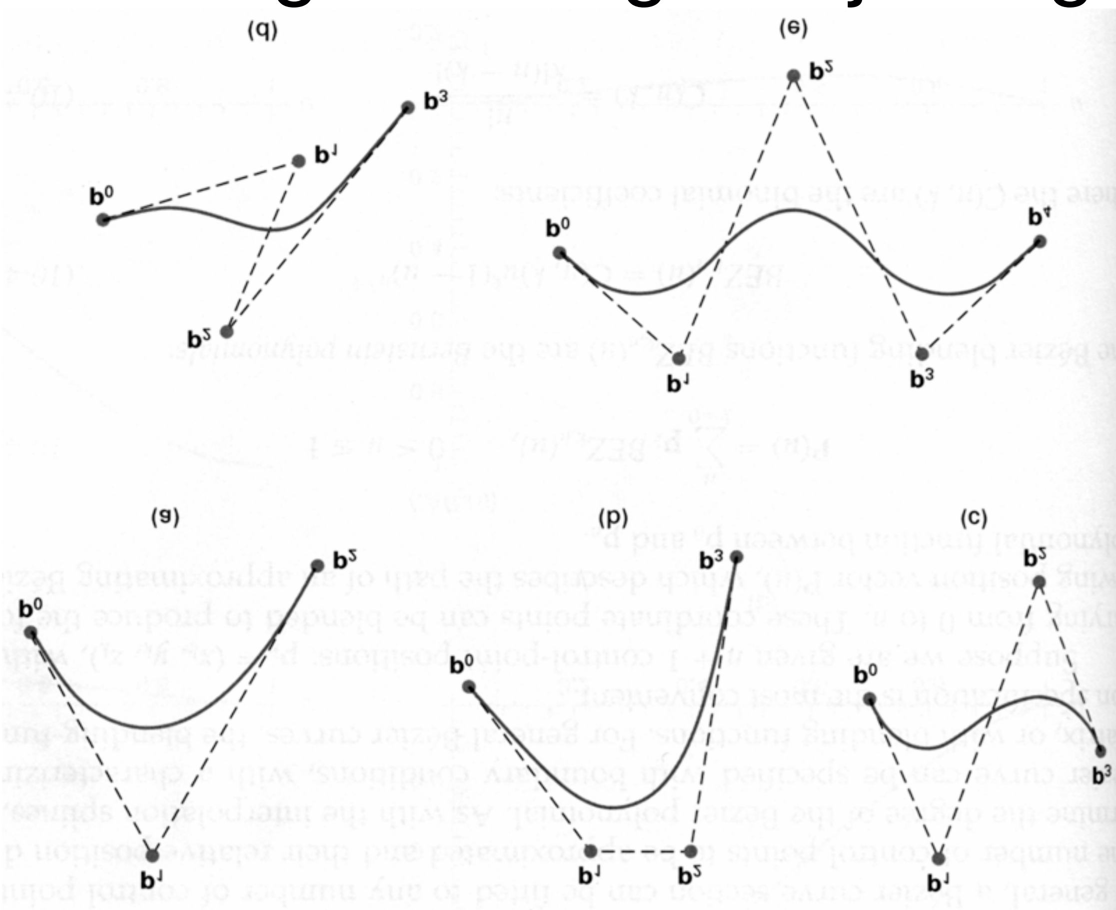
Bézier Curves

- curve will always remain within convex hull (bounding region) defined by control points



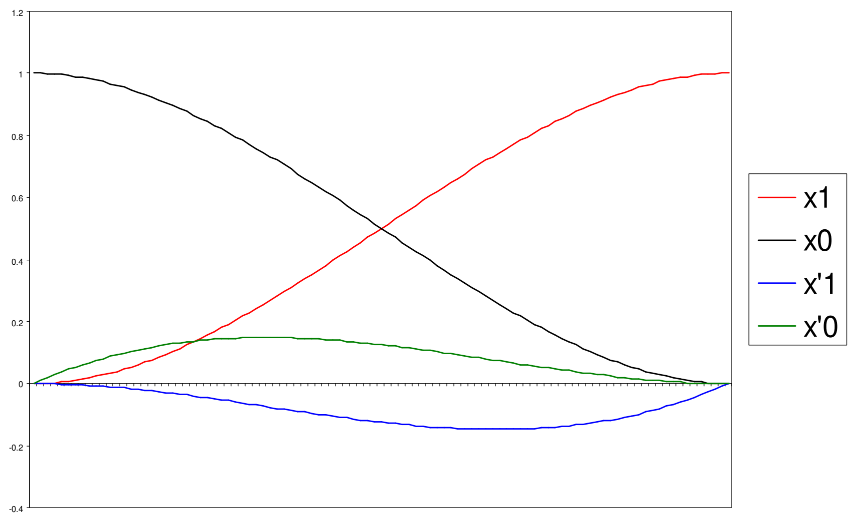
Bézier Curves

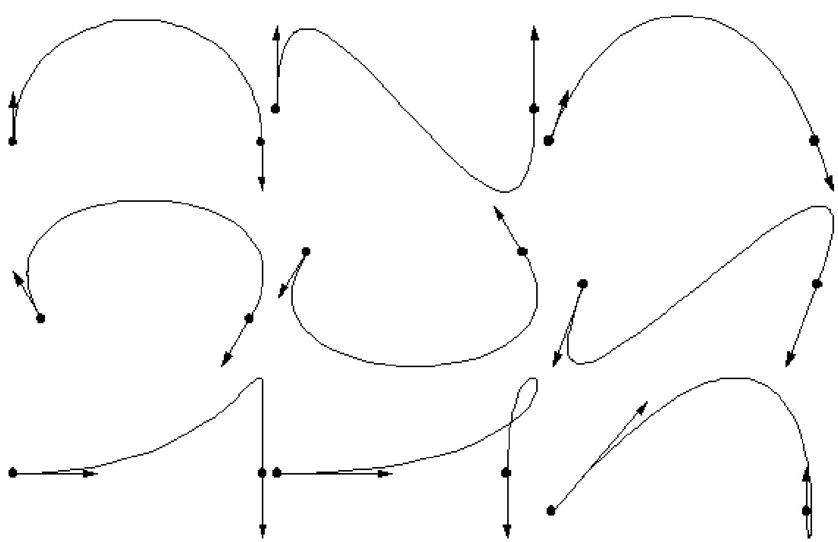
- interpolate between first, last control points
- 1^{st} point's tangent along line joining 1^{st}, 2^{nd} pts
- 4^{th} point's tangent along line joining 3^{rd}, 4^{th} pts



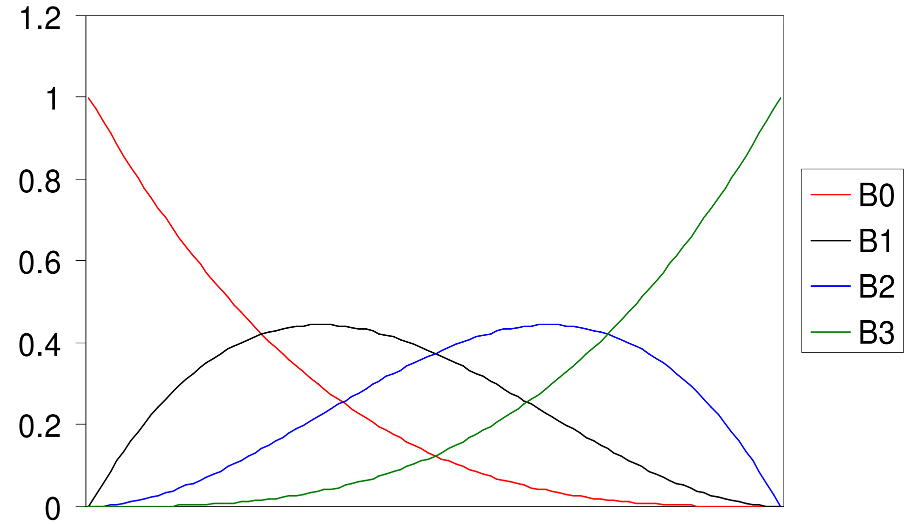
Comparing Hermite and Bezier

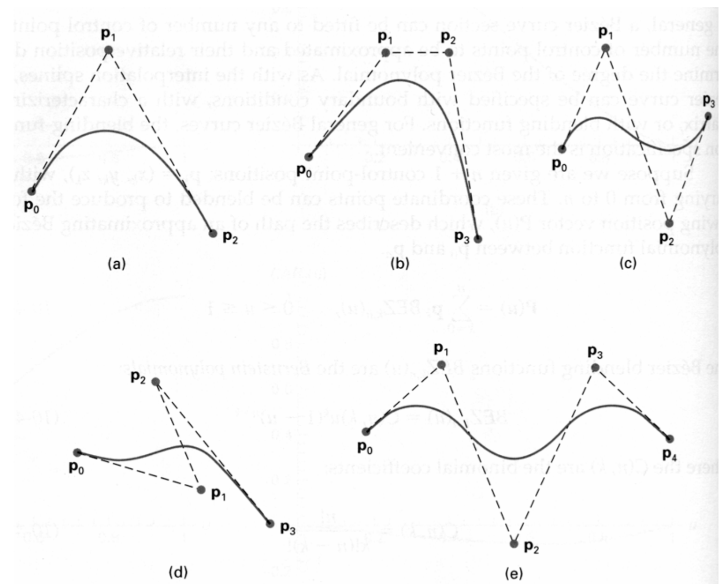
Hermite





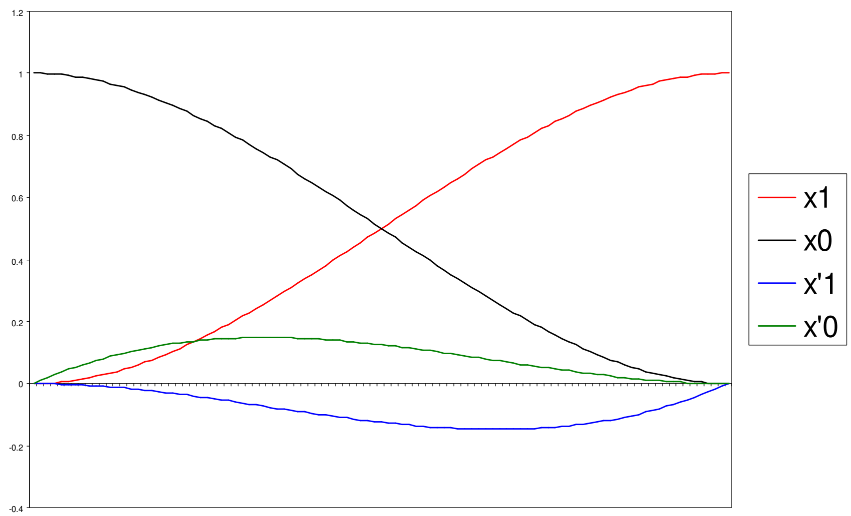
Bezier

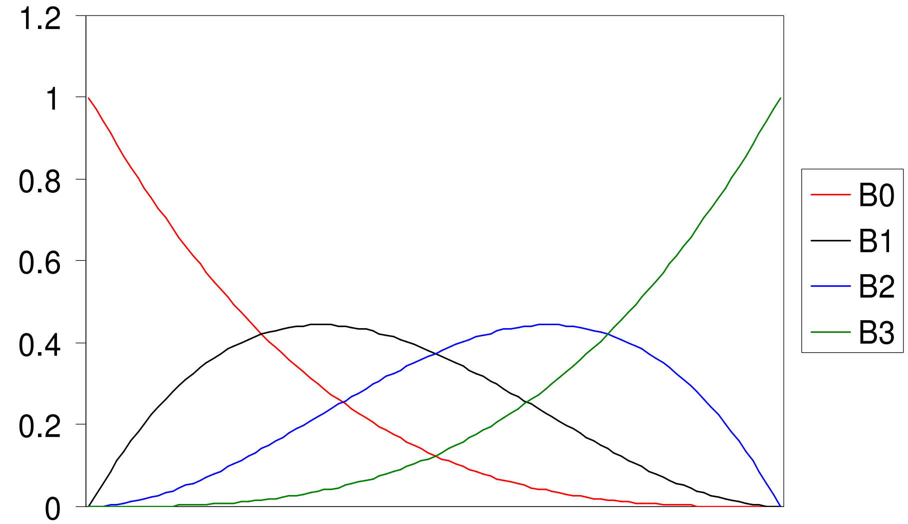


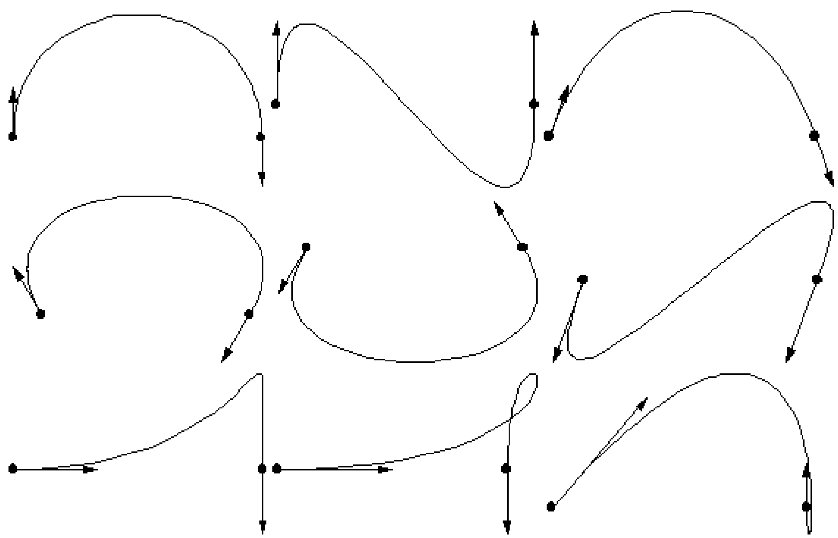


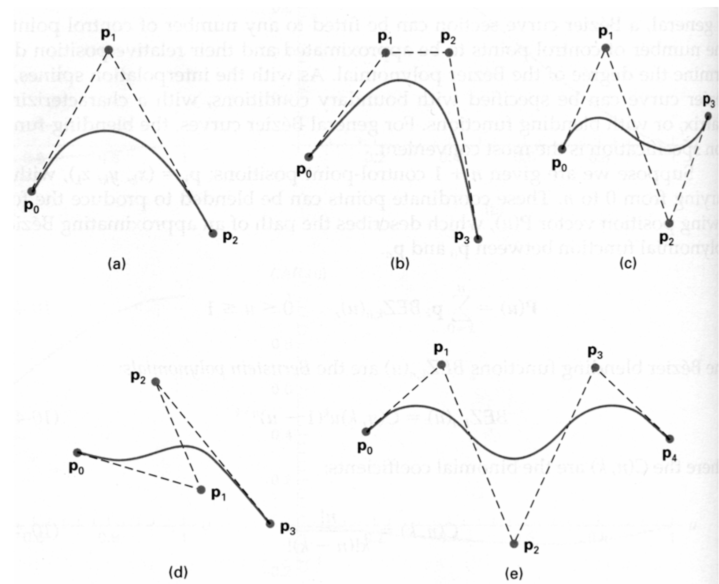
Comparing Hermite and Bezier

demo: www.siggraph.org/education/materials/HyperGraph/modeling/splines/demoprog/curve.html



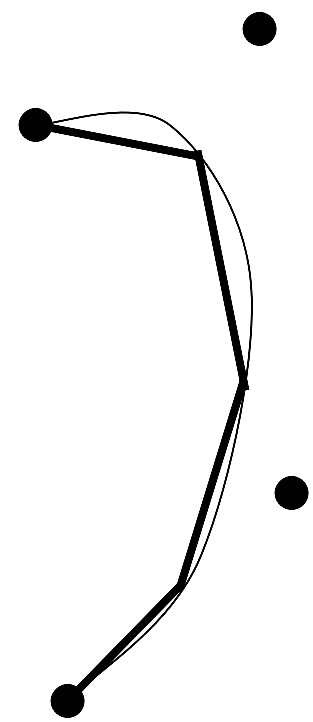






Rendering Bezier Curves: Simple

- evaluate curve at fixed set of parameter values, join points with straight lines
- advantage: very simple
- disadvantages:
  - expensive to evaluate the curve at many points
  - no easy way of knowing how fine to sample points, and maybe sampling rate must be different along curve
  - no easy way to adapt: hard to measure deviation of line segment from exact curve

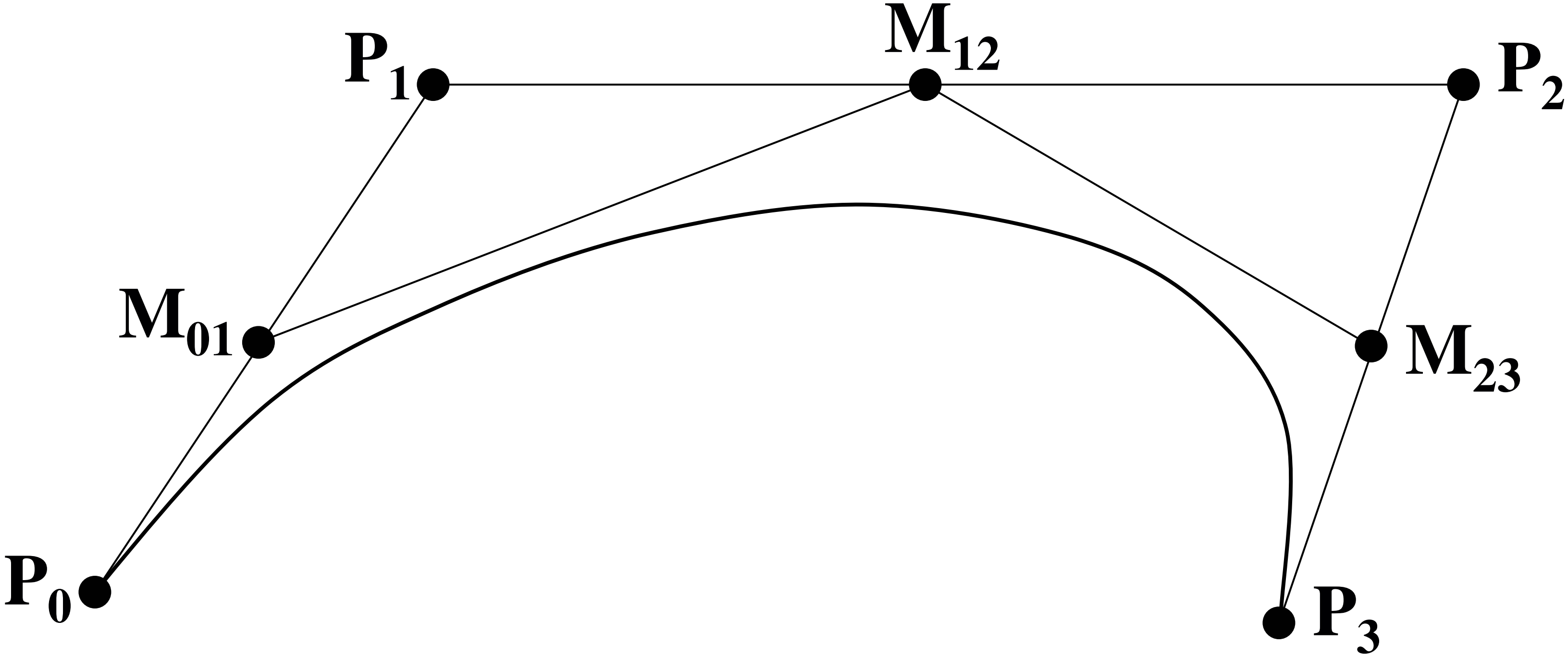


Rendering Beziers: Subdivision

- a cubic Bezier curve can be broken into two shorter cubic Bezier curves that exactly cover original curve
- suggests a rendering algorithm:
  - keep breaking curve into sub-curves
  - stop when control points of each sub-curve are nearly collinear
  - draw the control polygon: polygon formed by control points

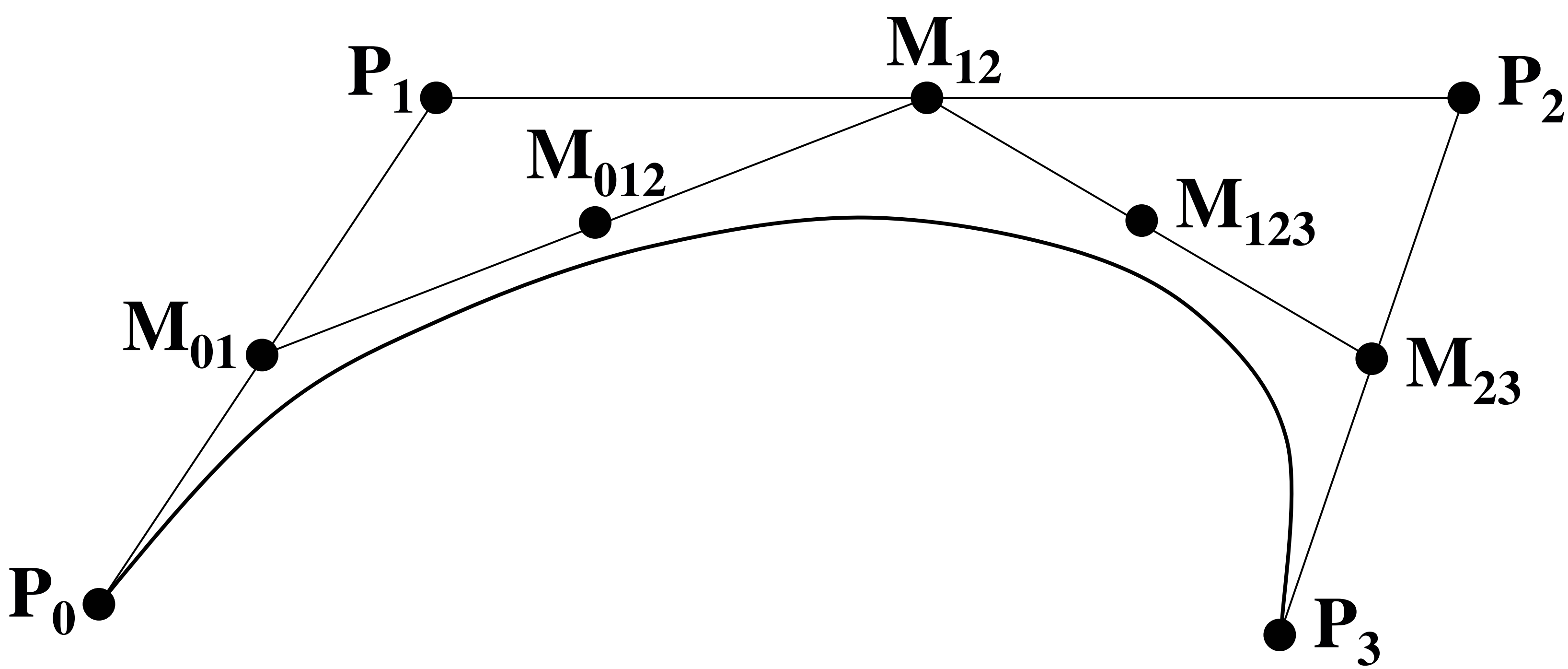
Sub-Dividing Bezier Curves

- step 1: find the midpoints of the lines joining the original control vertices. call them M_{01}, M_{12}, M_{23}



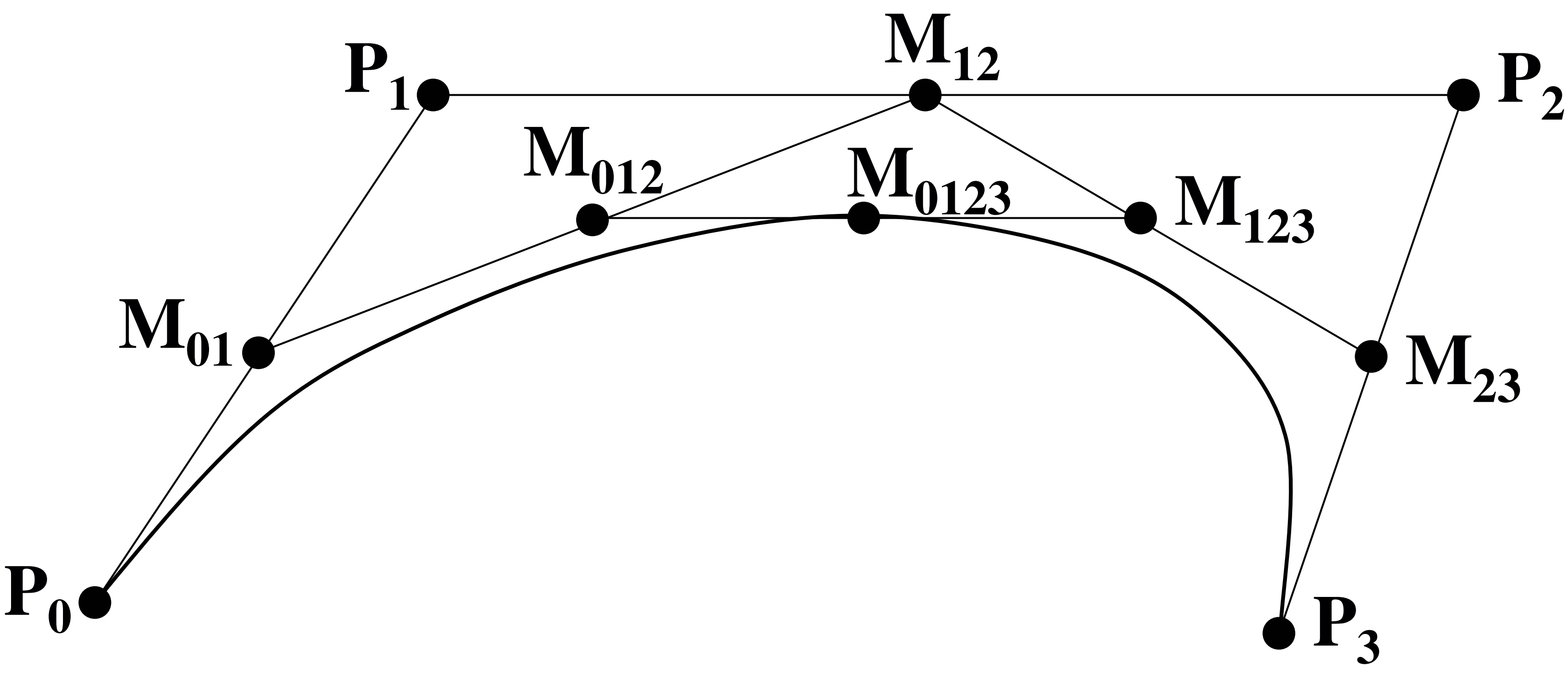
Sub-Dividing Bezier Curves

- step 2: find the midpoints of the lines joining M_{01}, M_{12} and M_{12}, M_{23}. call them M_{012}, M_{123}



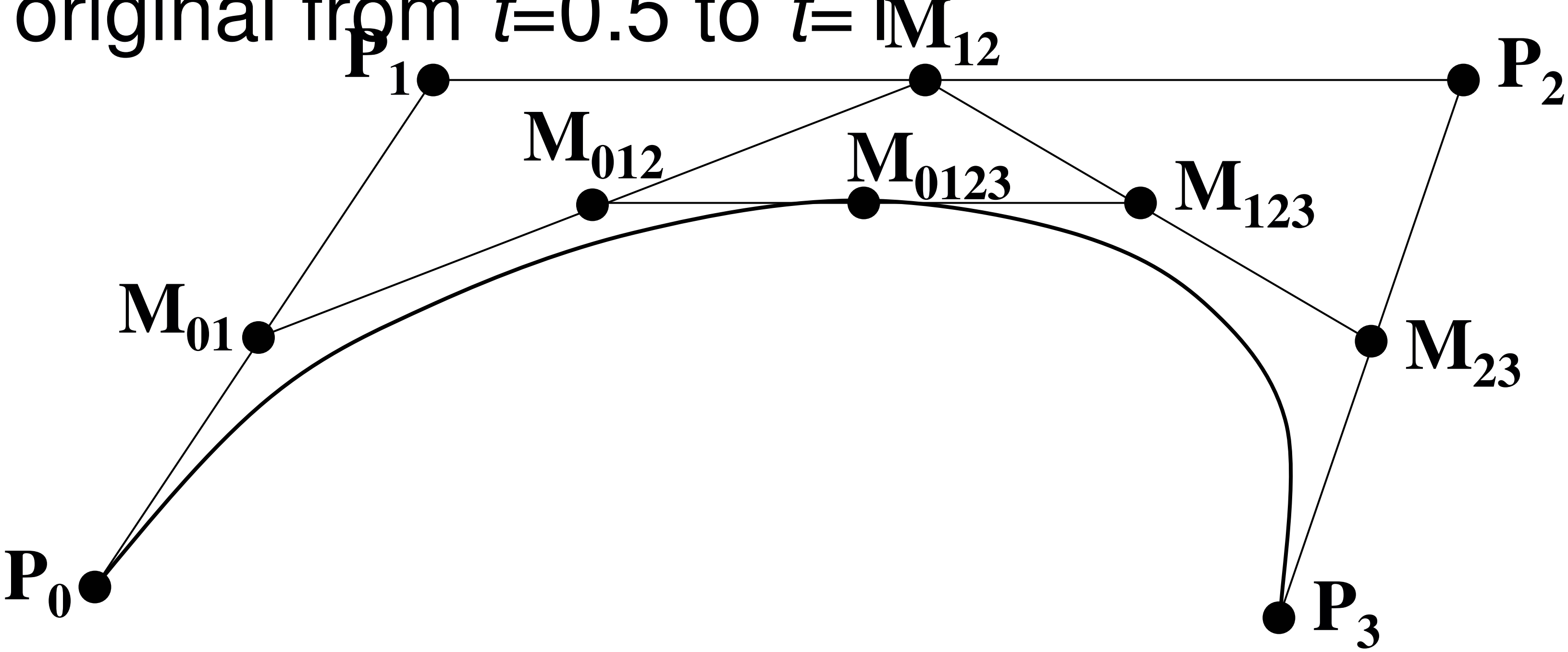
Sub-Dividing Bezier Curves

- step 3: find the midpoint of the line joining M_{012}, M_{123}. call it M_{0123}



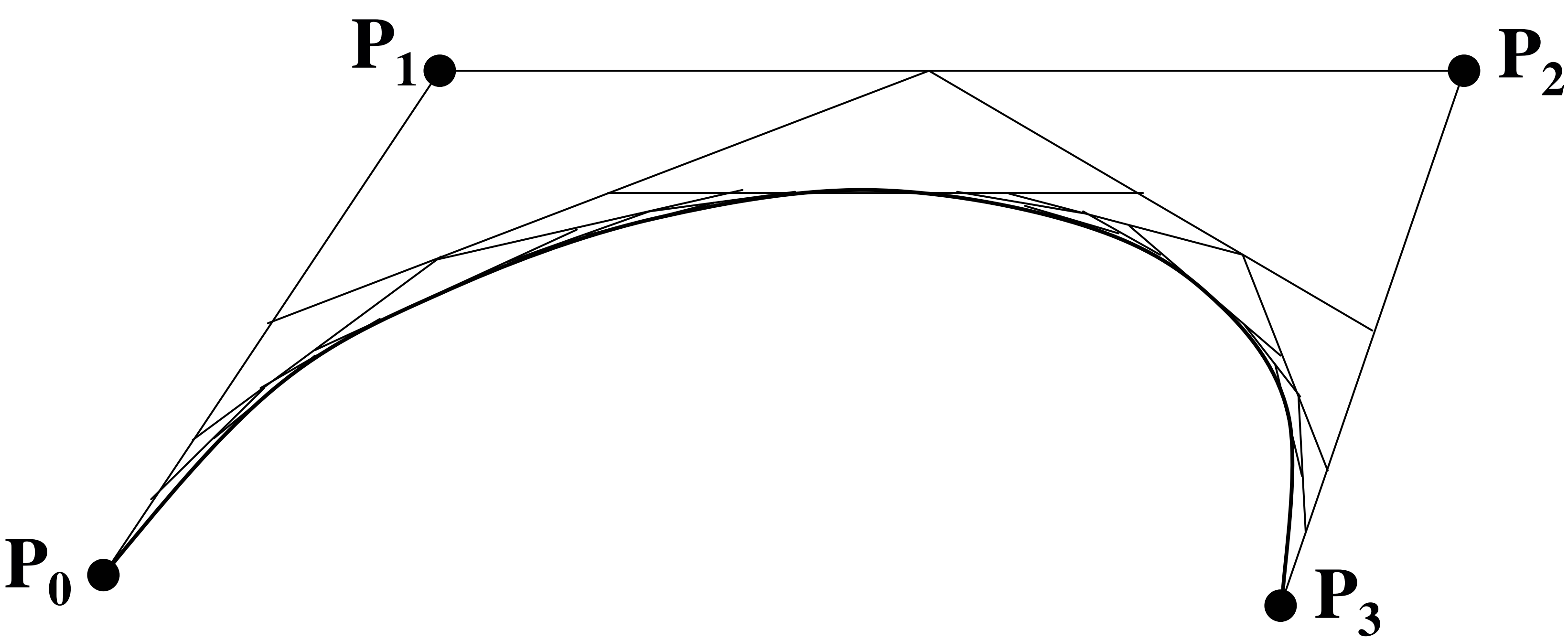
Sub-Dividing Bezier Curves

- curve P_0, M_{01}, M_{012}, M_{0123} exactly follows original from t=0 to t=0.5
- curve M_{0123} , M_{123} , M_{23}, P_3 exactly follows original from t=0.5 to t=1



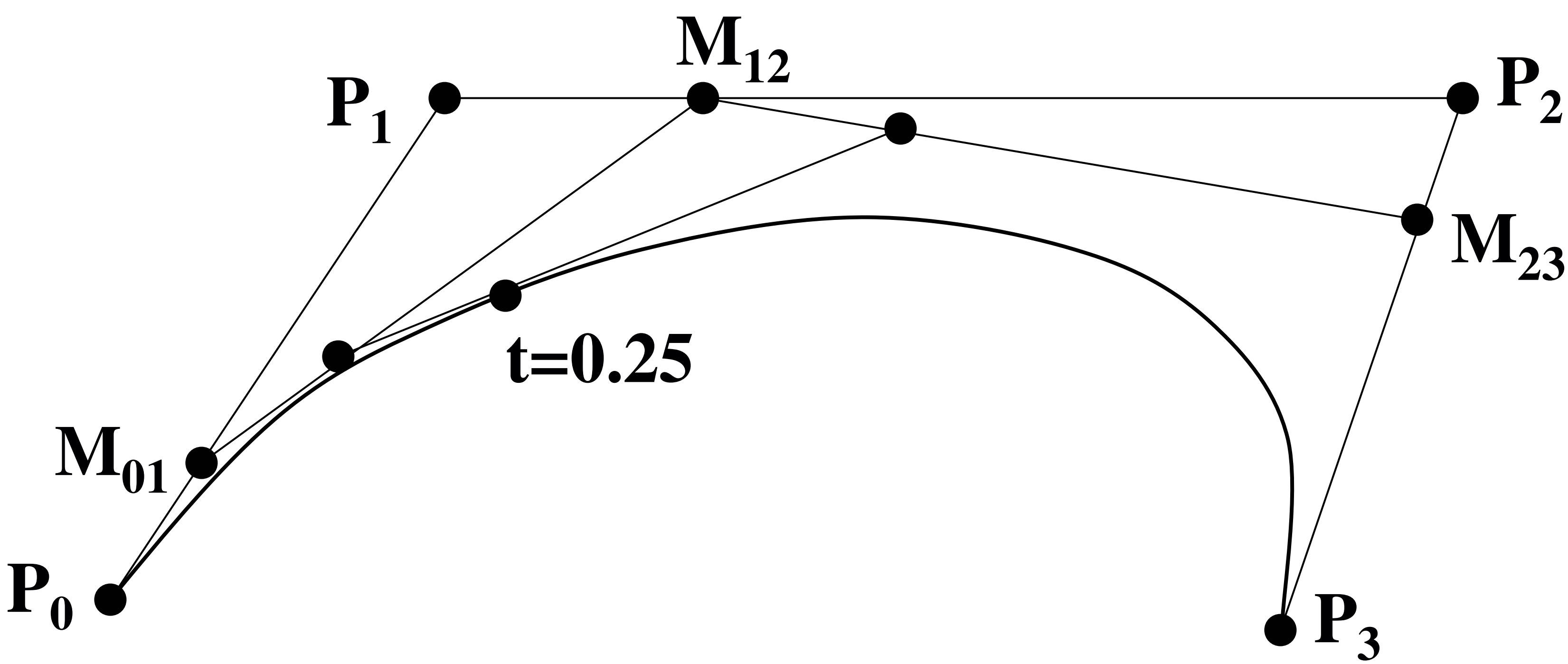
Sub-Dividing Bezier Curves

- continue process to create smooth curve



de Casteljau's Algorithm

- can find the point on a Bezier curve for any parameter value *t* with similar algorithm
  - for *t=0.25*, instead of taking midpoints take points 0.25 of the way

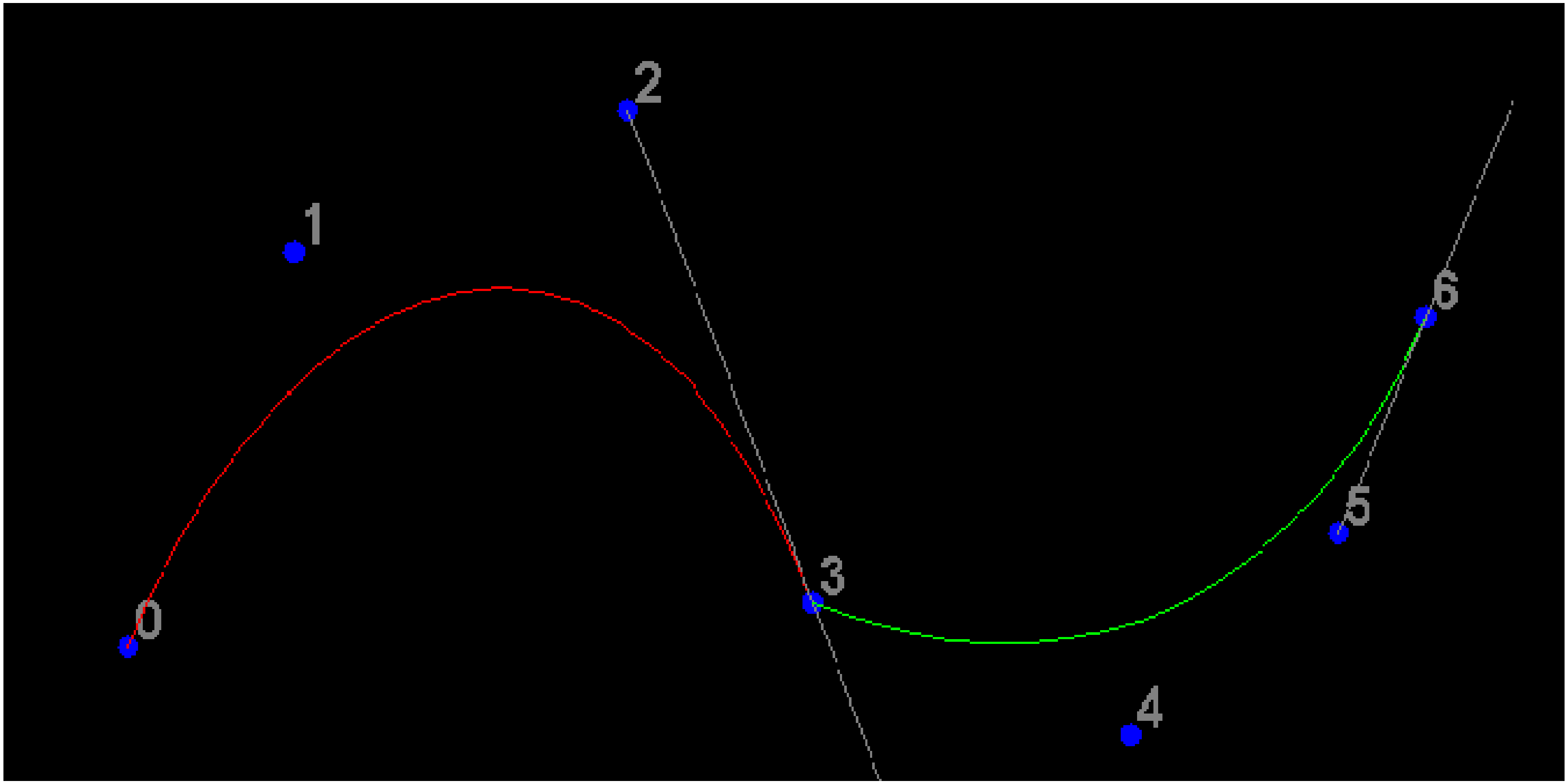


demo: www.saltire.com/applets/advanced_geometry/spline/spline.htm

Longer Curves

- a single cubic Bezier or Hermite curve can only capture a small class of curves
  - at most 2 inflection points
- one solution is to raise the degree
  - allows more control, at the expense of more control points and higher degree polynomials
  - control is not *local*, one control point influences entire curve
- better solution is to join pieces of cubic curve together into *piecewise cubic curves*
  - total curve can be broken into pieces, each of which is cubic
  - *local control*: each control point only influences a limited part of the curve
  - interaction and design is much easier

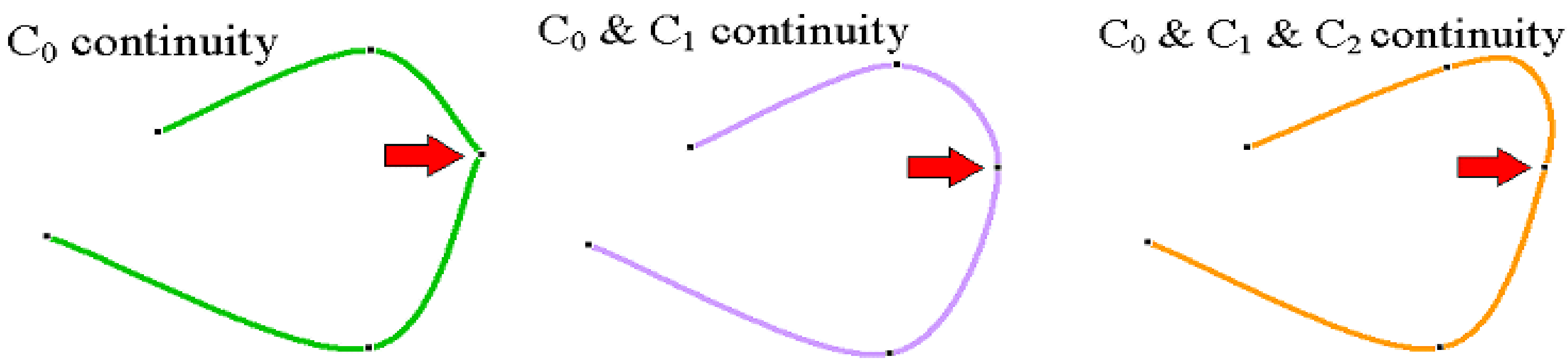
Piecewise Bezier: Continuity Problems



demo: www.cs.princeton.edu/~min/cs426/jar/bezier.html

Continuity

- when two curves joined, typically want some degree of continuity across knot boundary
  - C^0, “C-zero”, point-wise continuous, curves share same point where they join
  - C^1, “C-one”, continuous derivatives
  - C^2, “C-two”, continuous second derivatives



Geometric Continuity

- derivative continuity is important for animation
  - if object moves along curve with constant parametric speed, should be no sudden jump at knots
- for other applications, *tangent continuity* suffices
  - requires that the tangents point in the same direction
  - referred to as G^1 *geometric continuity*
  - curves could be made C^1 with a re-parameterization
  - geometric version of C^2 is G^2, based on curves having the same radius of curvature across the knot

Achieving Continuity

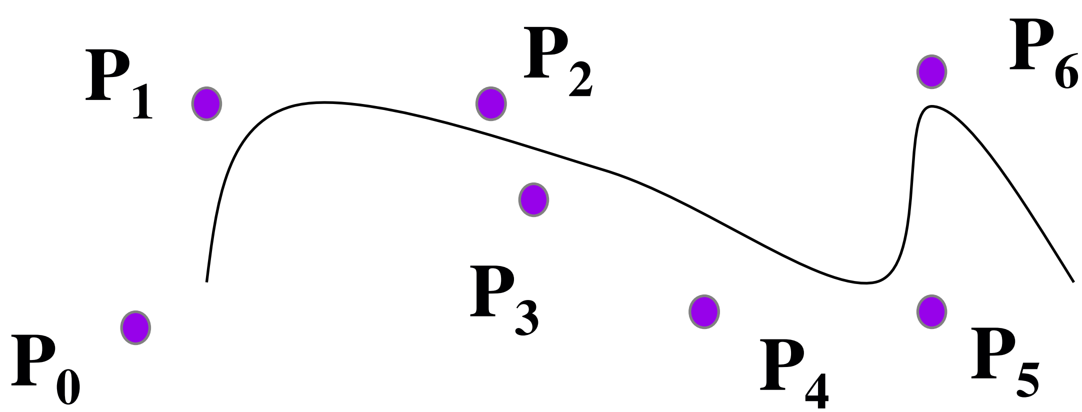
- Hermite curves
  - user specifies derivatives, so C^1 by sharing points and derivatives across knot
- Bezier curves
  - they interpolate endpoints, so C^0 by sharing control pts
  - introduce additional constraints to get C^1
    - parametric derivative is a constant multiple of vector joining first/last 2 control points
    - so C^1 achieved by setting $P_{0,3}=P_{1,0}=J$, and making $P_{0,2}$ and J and $P_{1,1}$ collinear, with $J-P_{0,2}=P_{1,1}-J$
    - C^2 comes from further constraints on P and P

B-Spline Curve

- start with a sequence of control points
- select four from middle of sequence

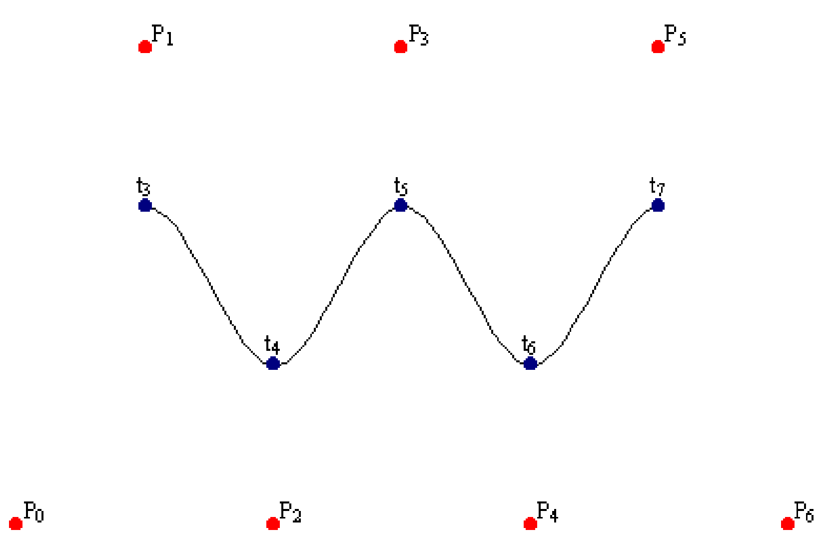
  (p_{i-2}, p_{i-1}, p_i, p_{i+1})

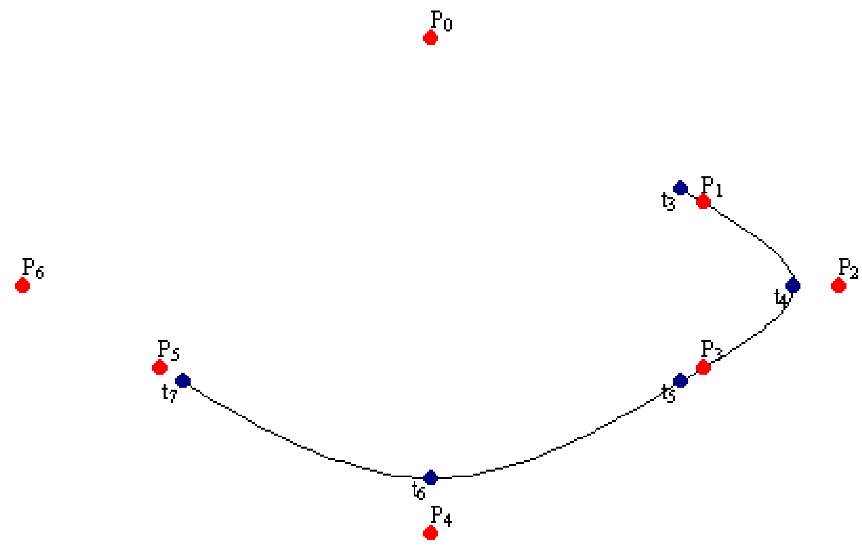
  – Bezier and Hermite goes between p_{i-2} and p_{i+1}
  – B-Spline doesn't interpolate (touch) any of them but approximates the going through p_{i-1} and p_i



B-Spline

- by far the most popular spline used
- C_0, C_1, and C_2 continuous





demo: www.siggraph.org/education/materials/HyperGraph/modeling/splines/demoprog/curve.html

B-Spline

- locality of points

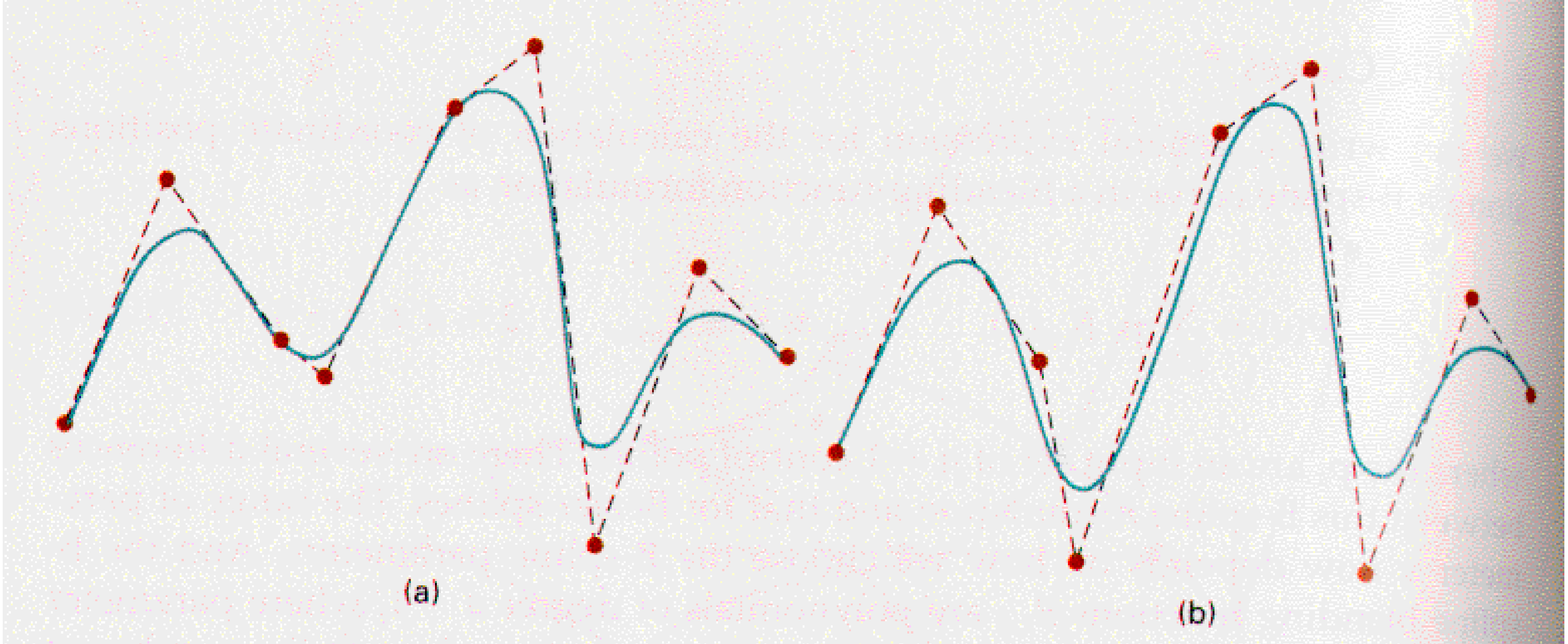


Figure 10-41
Local modification of a B-spline curve. Changing one of the control points in (a) produces curve (b), which is modified only in the neighborhood of the altered control point.