University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2007

Tamara Munzner

Curves

Week 12, Wed Apr 4

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007

Old News

- extra TA office hours in lab for hw/project Q&A
  - next week: Thu 4-6, Fri 10-2
  - last week of classes:
    - Mon 2-5, Tue 4-6, Wed 2-4, Thu 4-6, Fri 9-6
- final review Q&A session
  - Mon Apr 16 10-12
- reminder: no lecture/labs Fri 4/6, Mon 4/9

Old News

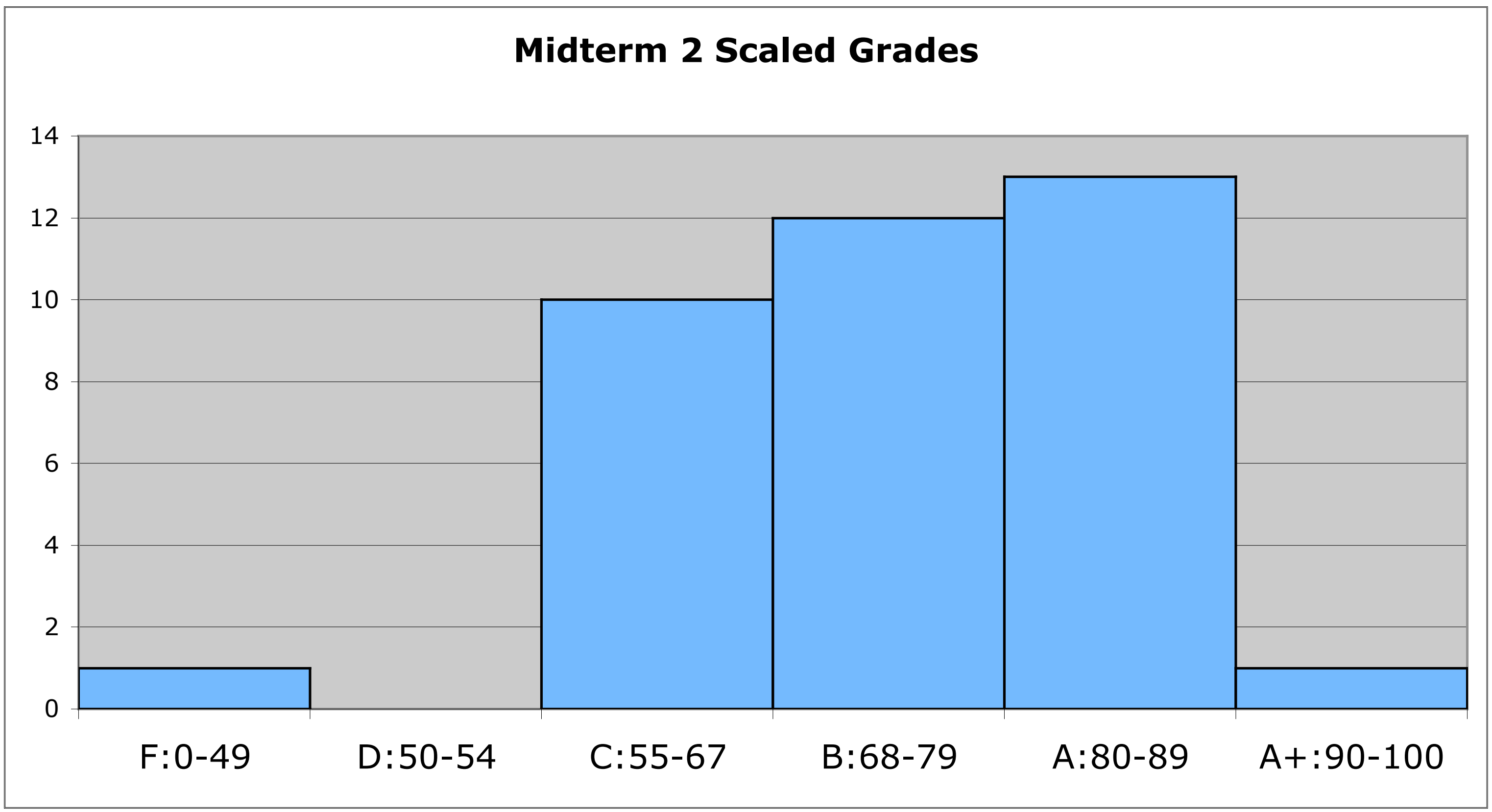
- project 4 grading slots signup
  - Wed Apr 18 10-12
  - Wed Apr 18 4-6
  - Fri Apr 20 10-1

Reminder for H4

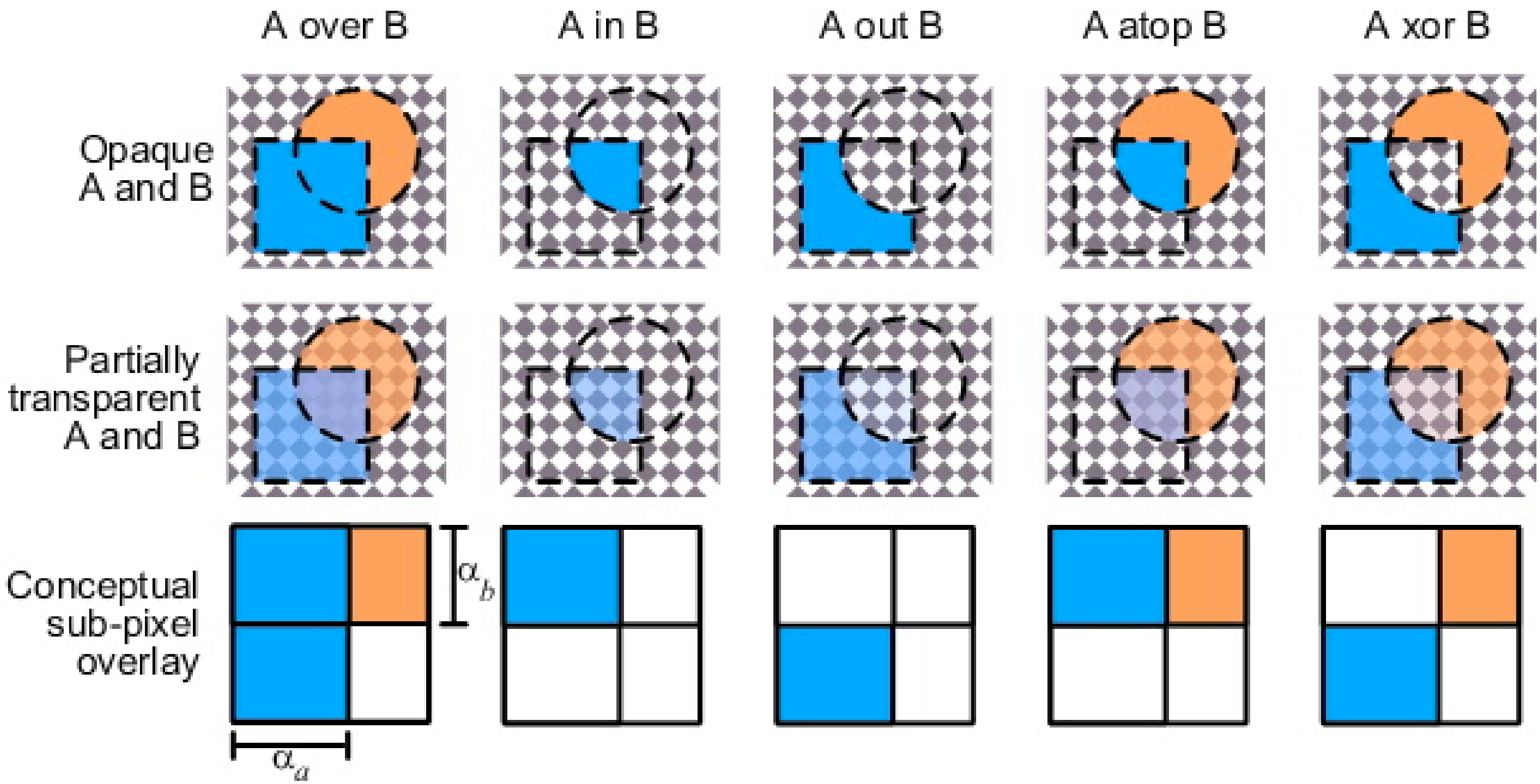
- For any answer involving calculation, although it's fine to show your work in analytical form, the final answer should be expressed as a number to two decimal places.

News

- regraded homeworks/exams handed back
- midterm handed back (scores are scaled)



Review: Compositing



Correction/Review: Premultiplying Colors

- specify opacity with alpha channel: (r,g,b,α)
  - α=1: opaque, α=.5: translucent, α=0: transparent

- **A** *over* **B**
  - $\mathbf{C} = \alpha\mathbf{A} + (1-\alpha)\mathbf{B}$

- but what if **B** is also partially transparent**?**
  - $\mathbf{C} = \alpha\mathbf{A} + (1-\alpha)\,\beta\mathbf{B} = \beta\mathbf{B} + \alpha\mathbf{A}\cancel{+\beta\mathbf{B}} - \alpha\,\beta\mathbf{B}$
  - $\gamma = \beta + (1-\beta)\alpha = \beta + \alpha - \alpha\beta$
    - 3 multiplies, different equations for alpha vs. RGB

- premultiplying by alpha
  - $\mathbf{C'} = \gamma\,\mathbf{C},\ \mathbf{B'} = \beta\mathbf{B},\ \mathbf{A'} = \alpha\mathbf{A}$
  - $\mathbf{C'} = \mathbf{B'} + \mathbf{A'} - \alpha\mathbf{B'}$
  - $\gamma = \beta + \alpha - \alpha\beta$
    - 1 multiply to find C, same equations for alpha and RGB

Review: Rendering Pipeline

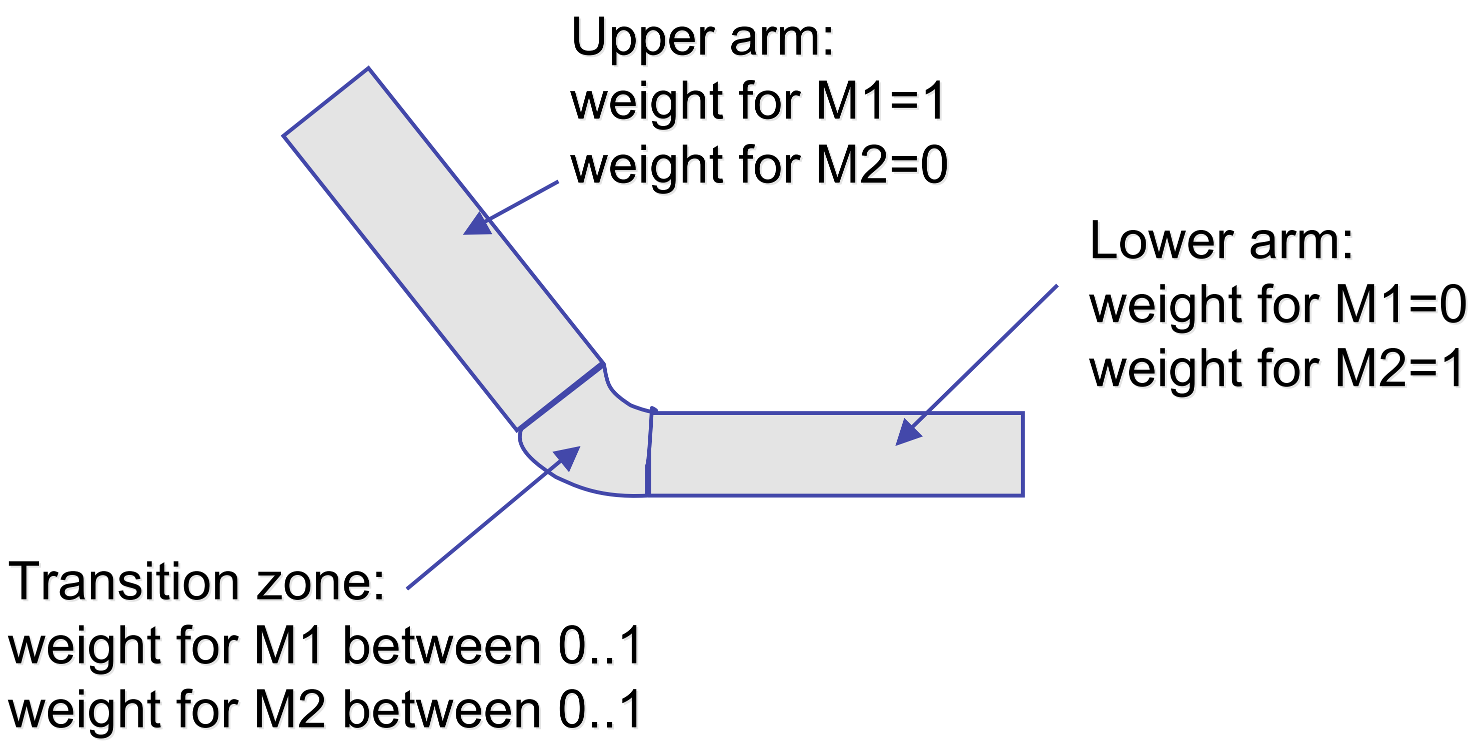
- so far rendering pipeline as a specific set of stages with **fixed functionality**
- modern graphics hardware more flexible
  - programmable “vertex shaders” replace several geometry processing stages
  - programmable “fragment/pixel shaders” replace texture mapping stage
  - hardware with these features now called Graphics Processing Unit (GPU)
- program shading hardware with assembly language analog, or high level shading language

Review: Vertex Shaders

- replace model/view transformation, lighting, perspective projection
- a little assembly-style program is executed on every individual vertex independently
- it sees:
  - vertex attributes that change per vertex:
    - position, color, texture coordinates…
  - registers that are constant for all vertices (changes are expensive):
    - matrices, light position and color, …
  - temporary registers
  - output registers for position, color, tex coords…

Review: Skinning Vertex Shader

- arm example:
  - M1: matrix for upper arm
  - M2: matrix for lower arm



Review: Fragment Shaders

- fragment shaders operate on fragments in place of texturing hardware
  - after rasterization
  - before any fragment tests or blending
- input: fragment, with screen position, depth, color, and set of texture coordinates
- access to textures, some constant data, registers
- compute RGBA values for fragment, and depth
  - can also kill a fragment (throw it away)

Review: GPGPU Programming

- General Purpose GPU
  - use graphics card as SIMD parallel processor
  - textures as arrays
  - computation: render large quadrilateral
  - multiple rendering passes

Curves

Reading

- FCG Chap 13 Curves

Parametric Curves

- parametric form for a line:

$$x = x_0 t + (1-t)x_1$$

$$y = y_0 t + (1-t)y_1$$

$$z = z_0 t + (1-t)z_1$$

- x, y and z are each given by an equation that involves:
  - parameter t
  - some user specified control points, x_0 and x_1
- this is an example of a parametric curve

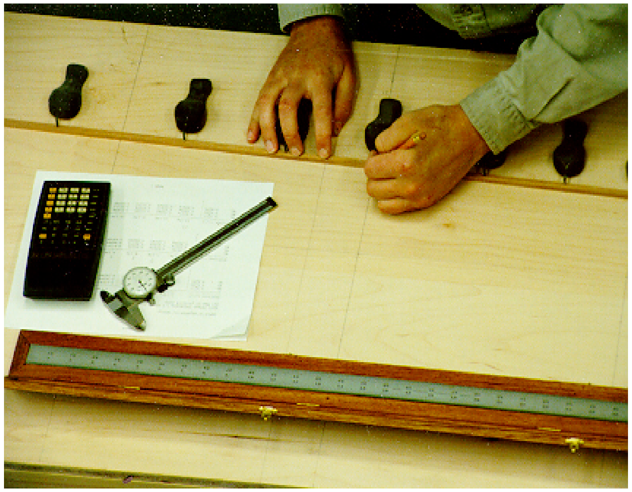
Splines

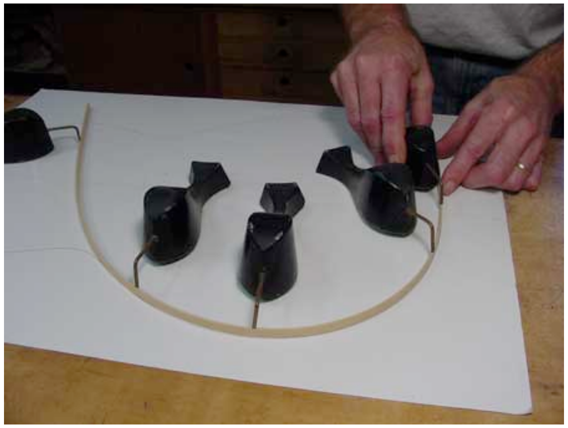
- a *spline* is a parametric curve defined by *control points*
  - term “spline” dates from engineering drawing, where a spline was a piece of flexible wood used to draw smooth curves
  - control points are *adjusted by the user* to control shape of curve

Splines - History

- draftsman used ‘ducks’ and strips of wood (splines) to draw curves
- wood splines have second-order continuity, pass through the control points

a duck (weight)





ducks trace out curve

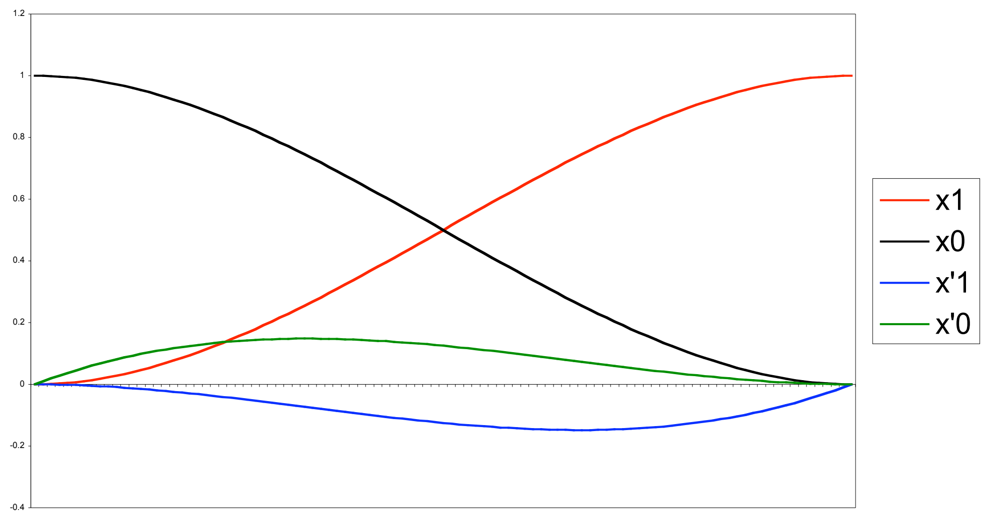
Hermite Spline

- *hermite spline* is curve for which user provides:
  - endpoints of curve
  - parametric derivatives of curve at endpoints
    - parametric derivatives are dx/dt, dy/dt, dz/dt

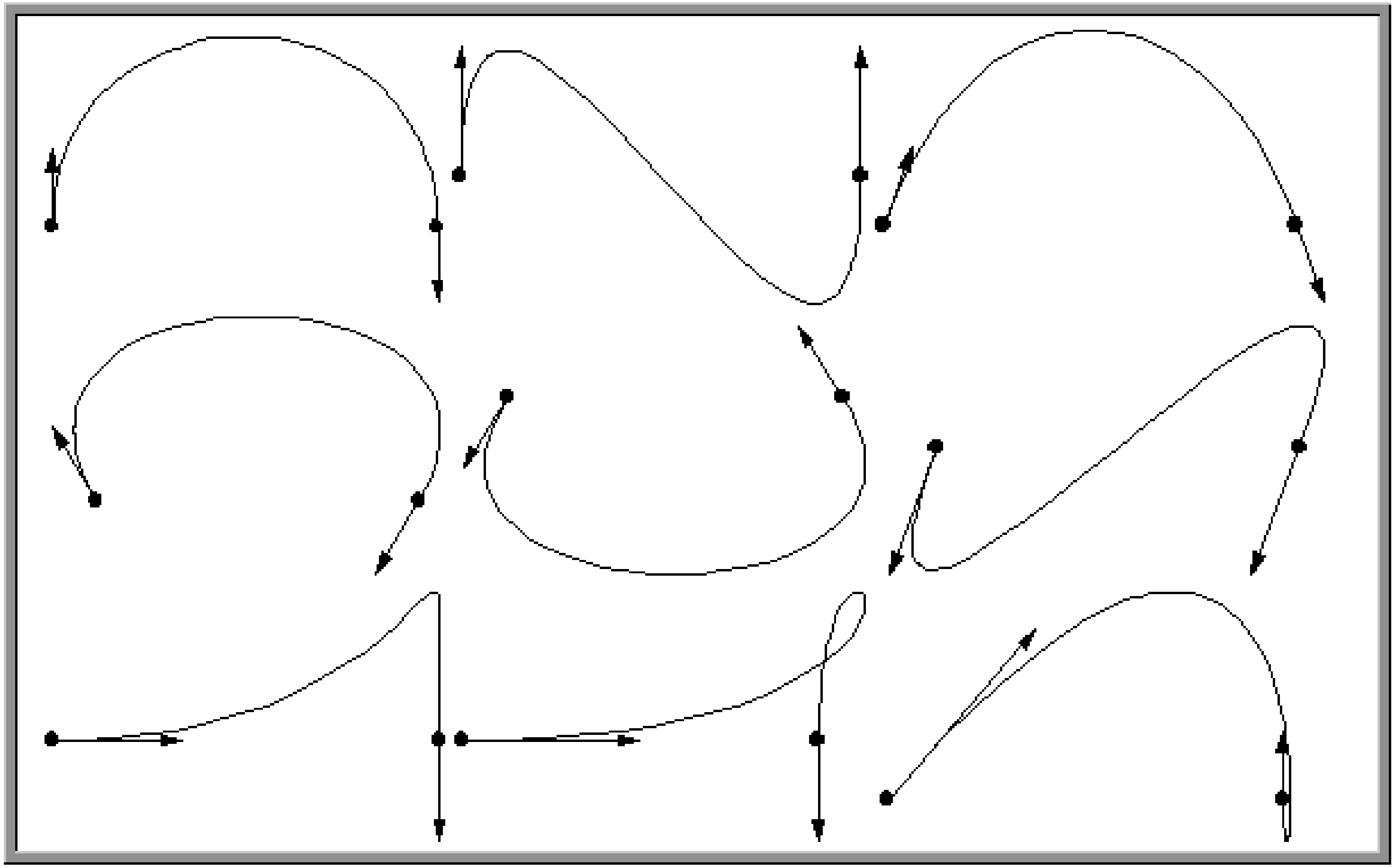
  - more derivatives would be required for higher order curves

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*

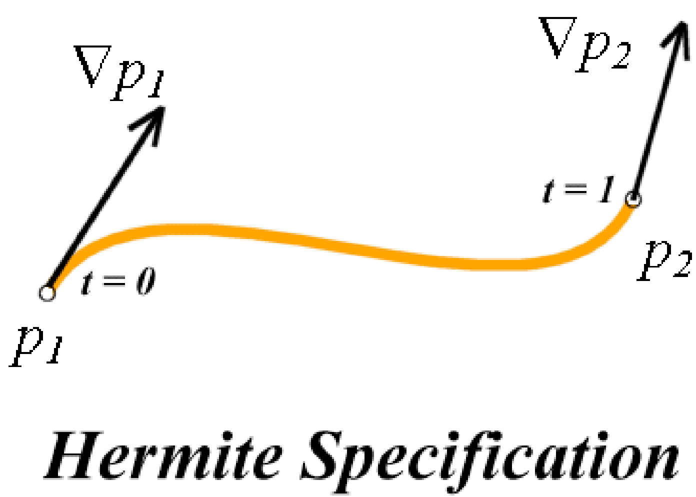


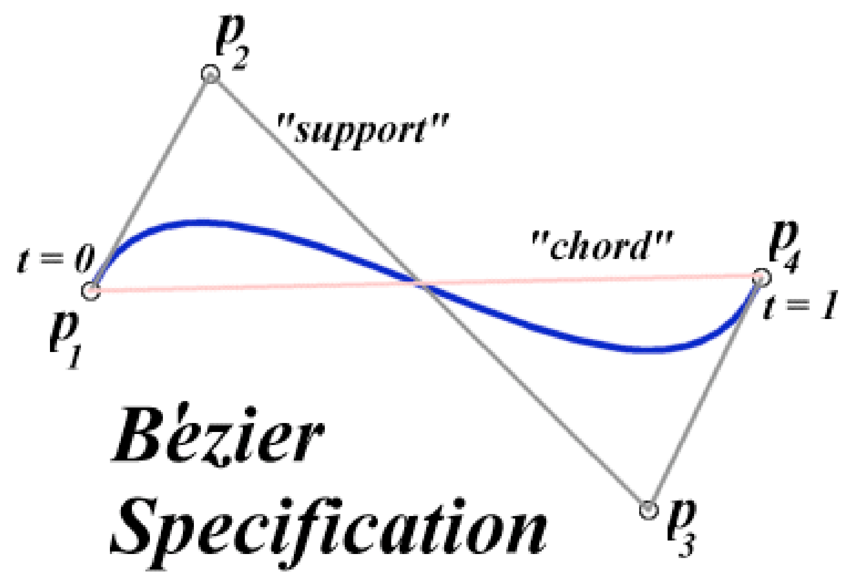
Sample Hermite Curves



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

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

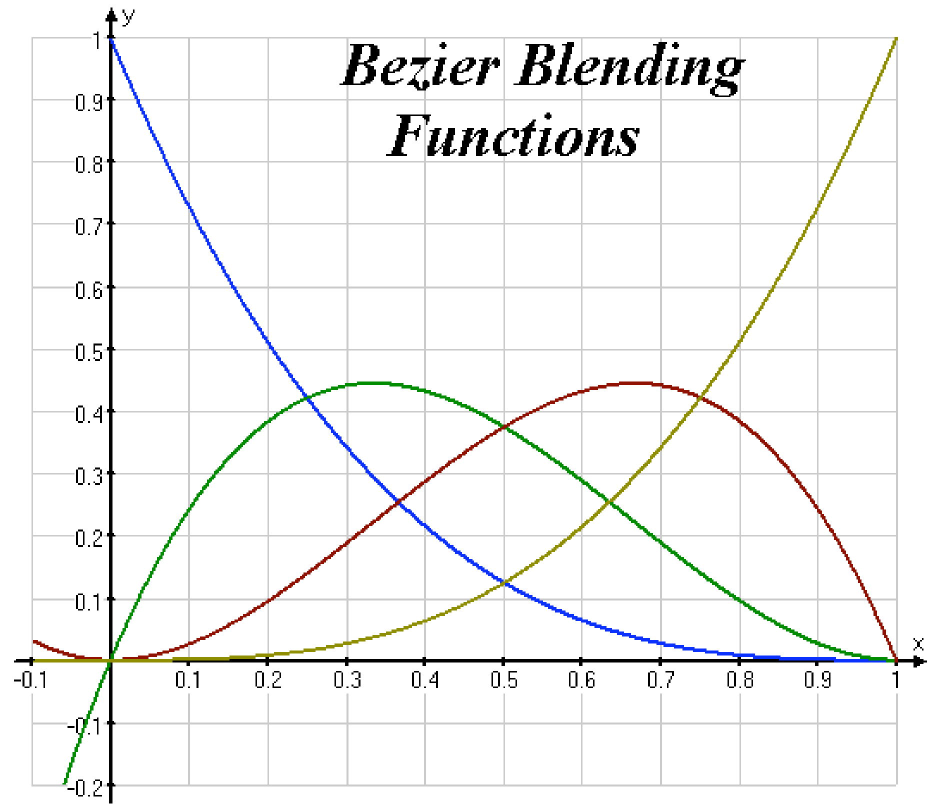
Bézier Blending Functions

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

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

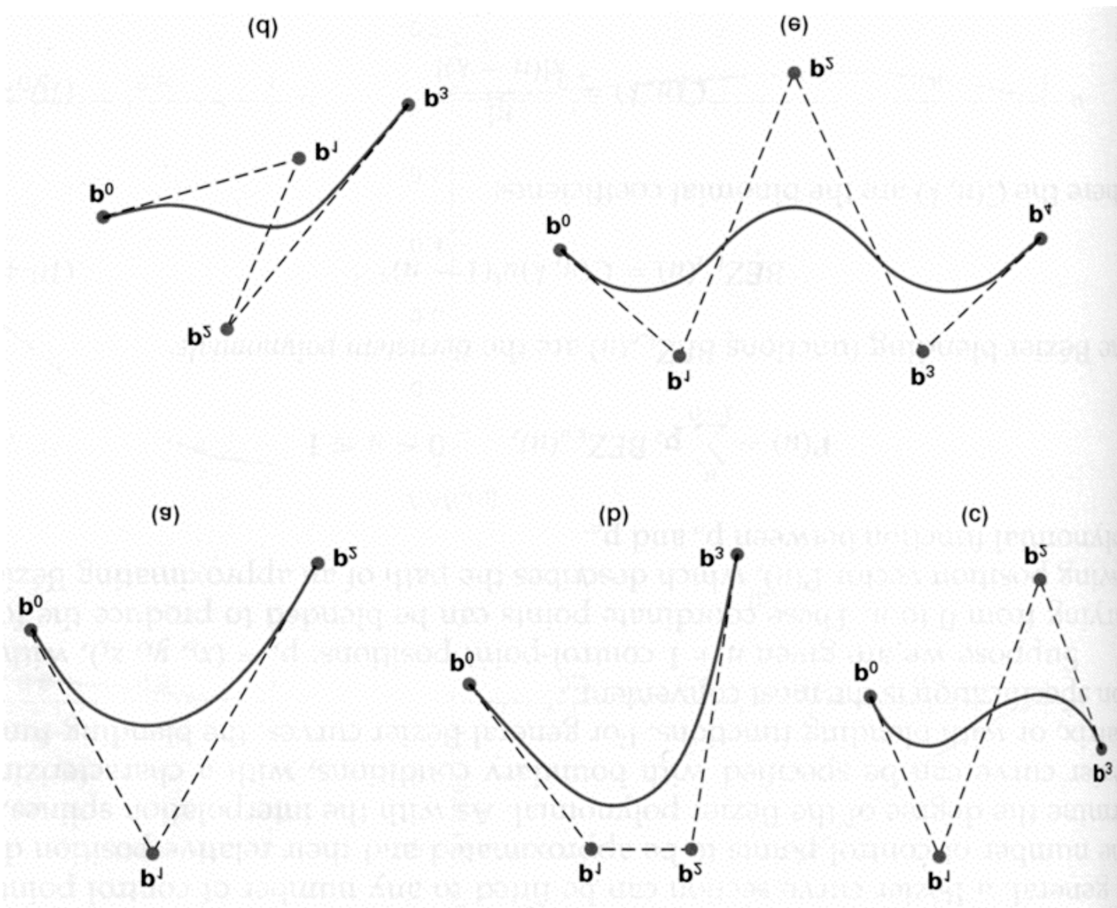
Bézier Blending Functions

- 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



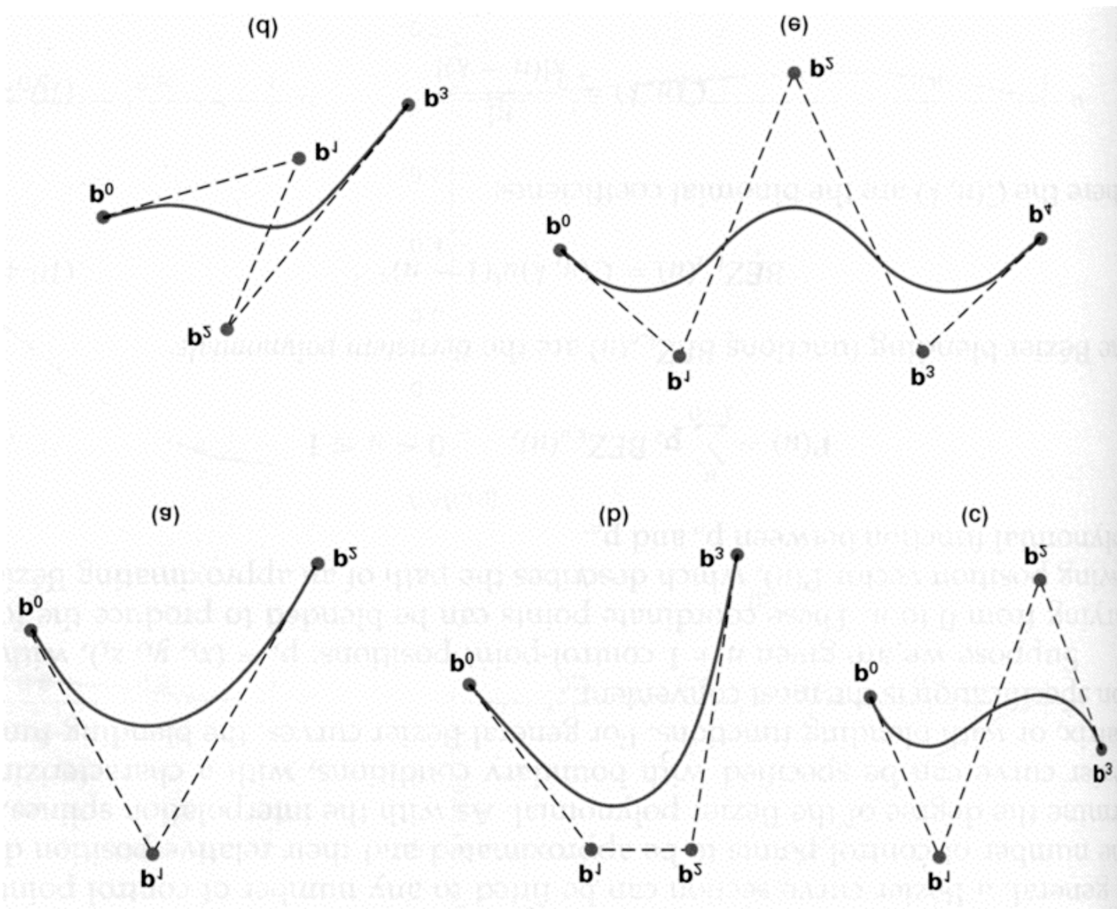
Bézier Curves

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



Bézier Curves

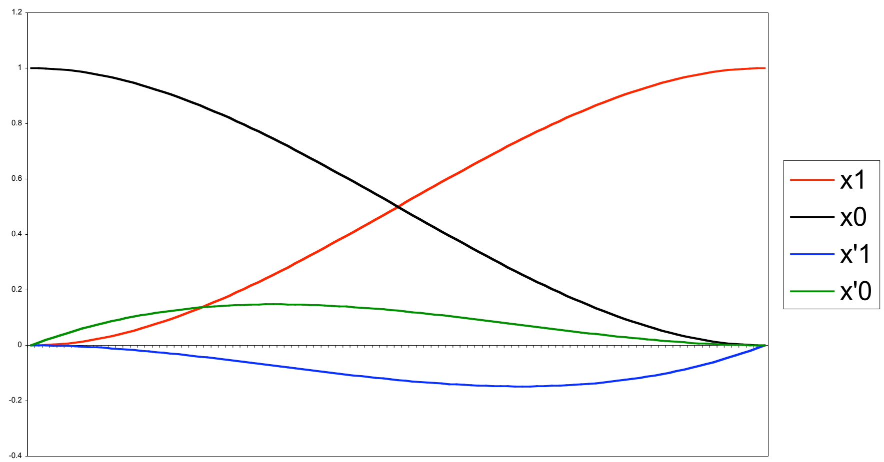
- interpolate between first, last control points
- 1st point's tangent along line joining 1st, 2nd pts
- 4th point's tangent along line joining 3rd, 4th pts

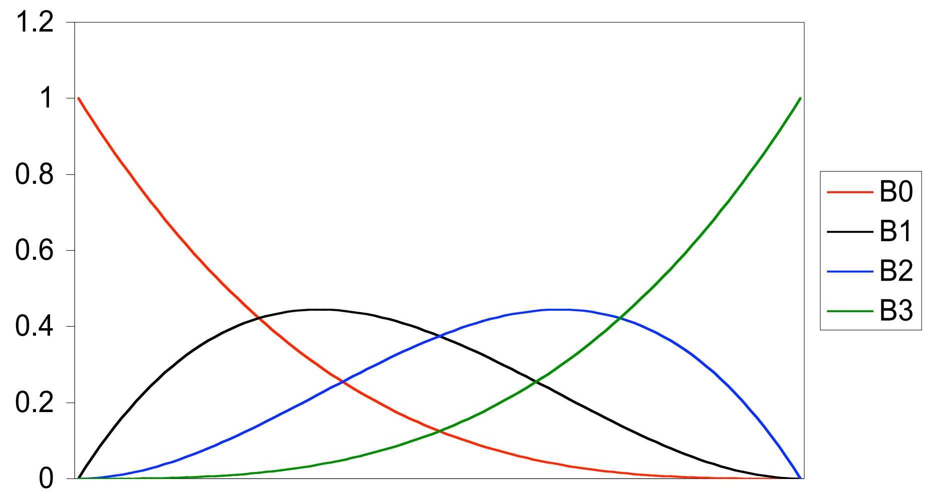


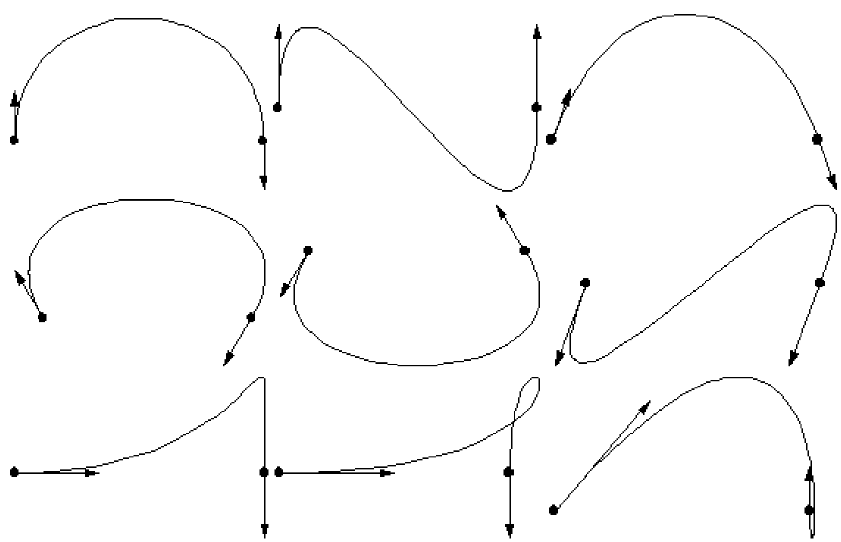
Comparing Hermite and Bézier

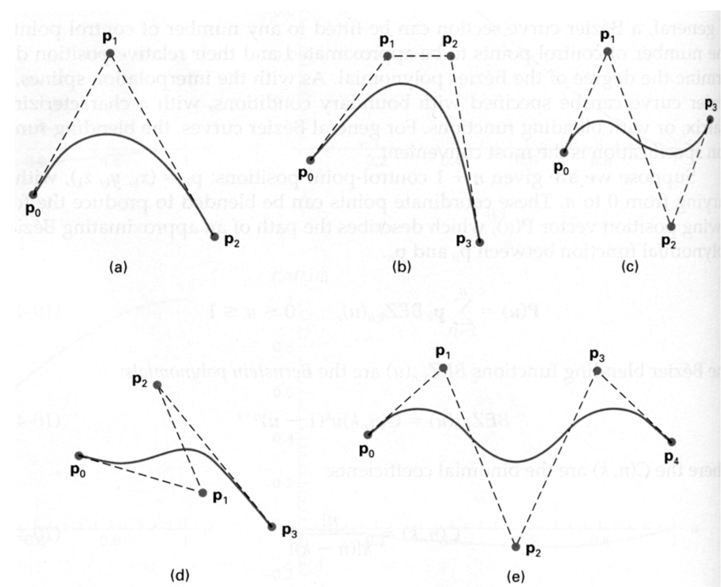
Hermite

Bézier



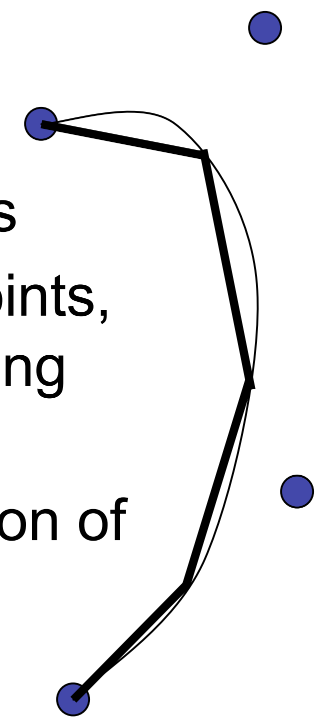






Rendering Bezier Curves: Simple

- 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

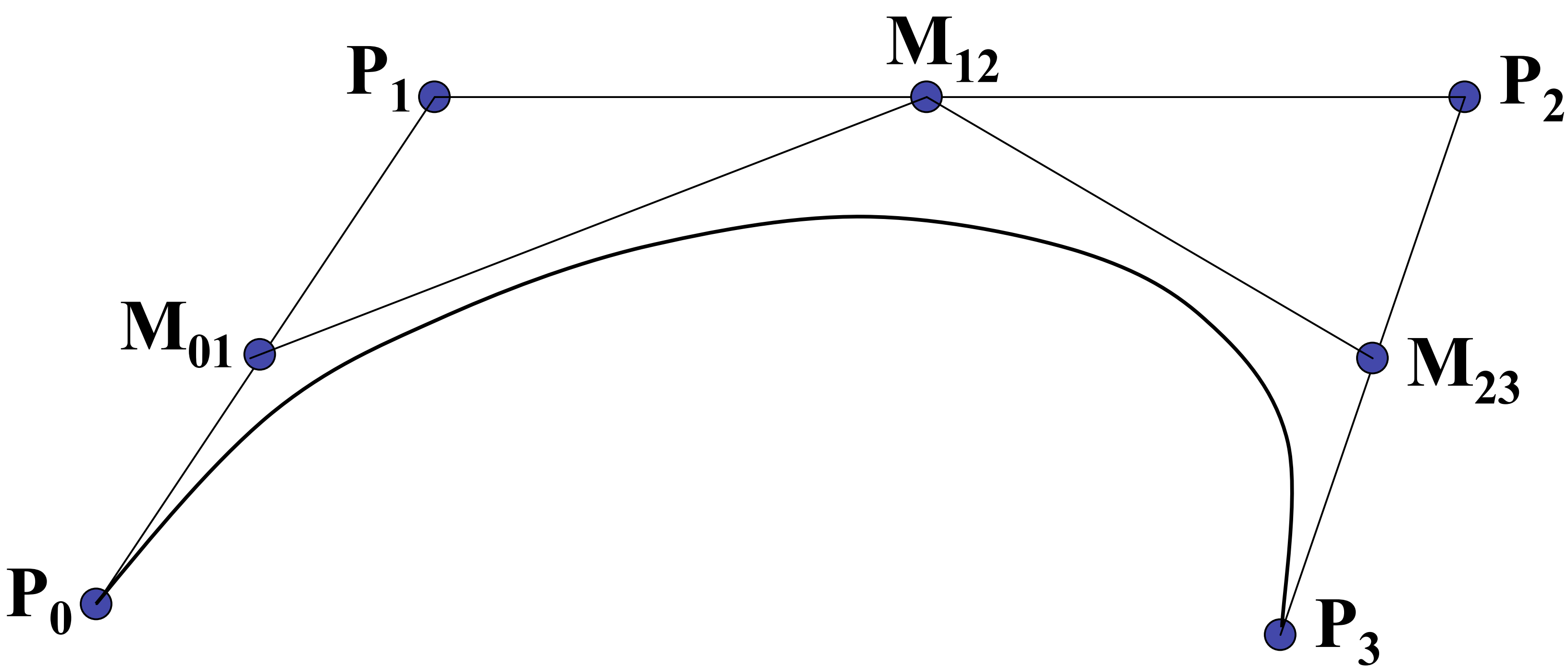


Rendering Beziers: Subdivision

- 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

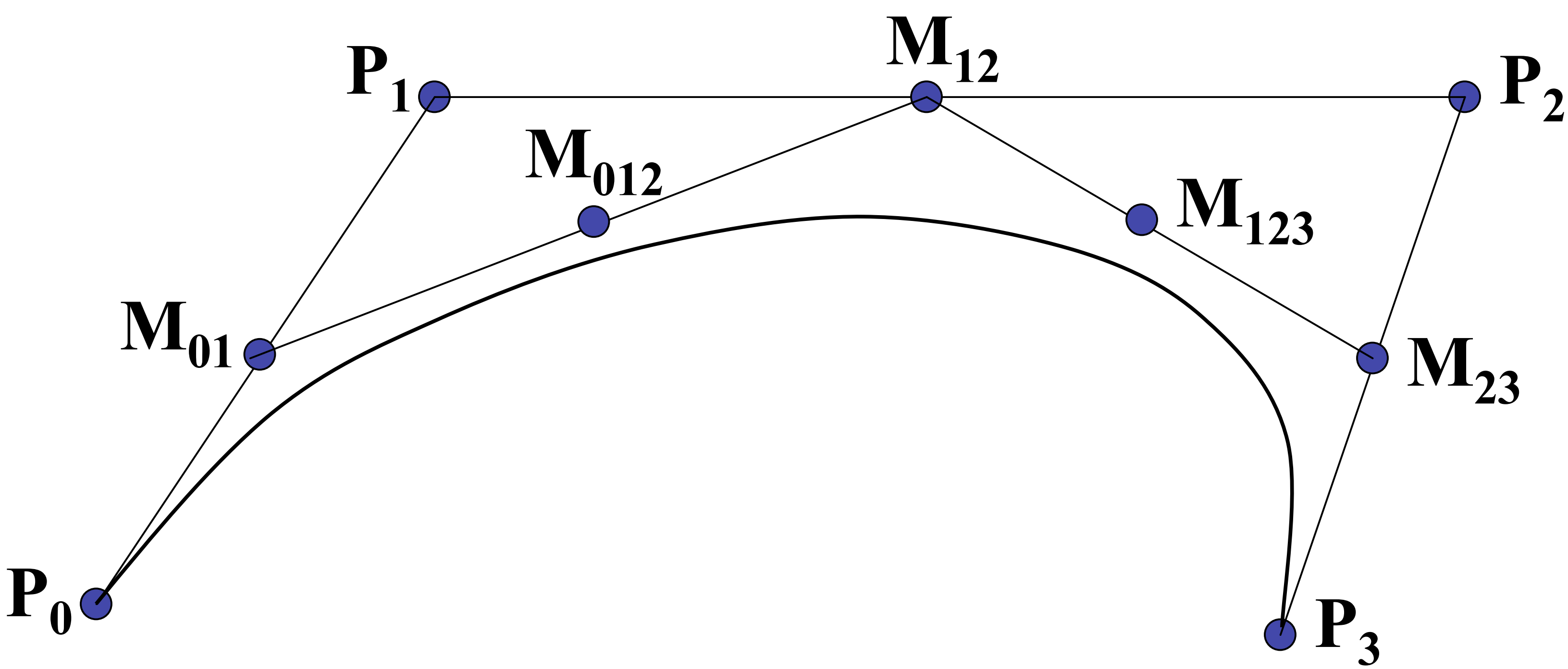
Sub-Dividing Bezier Curves

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



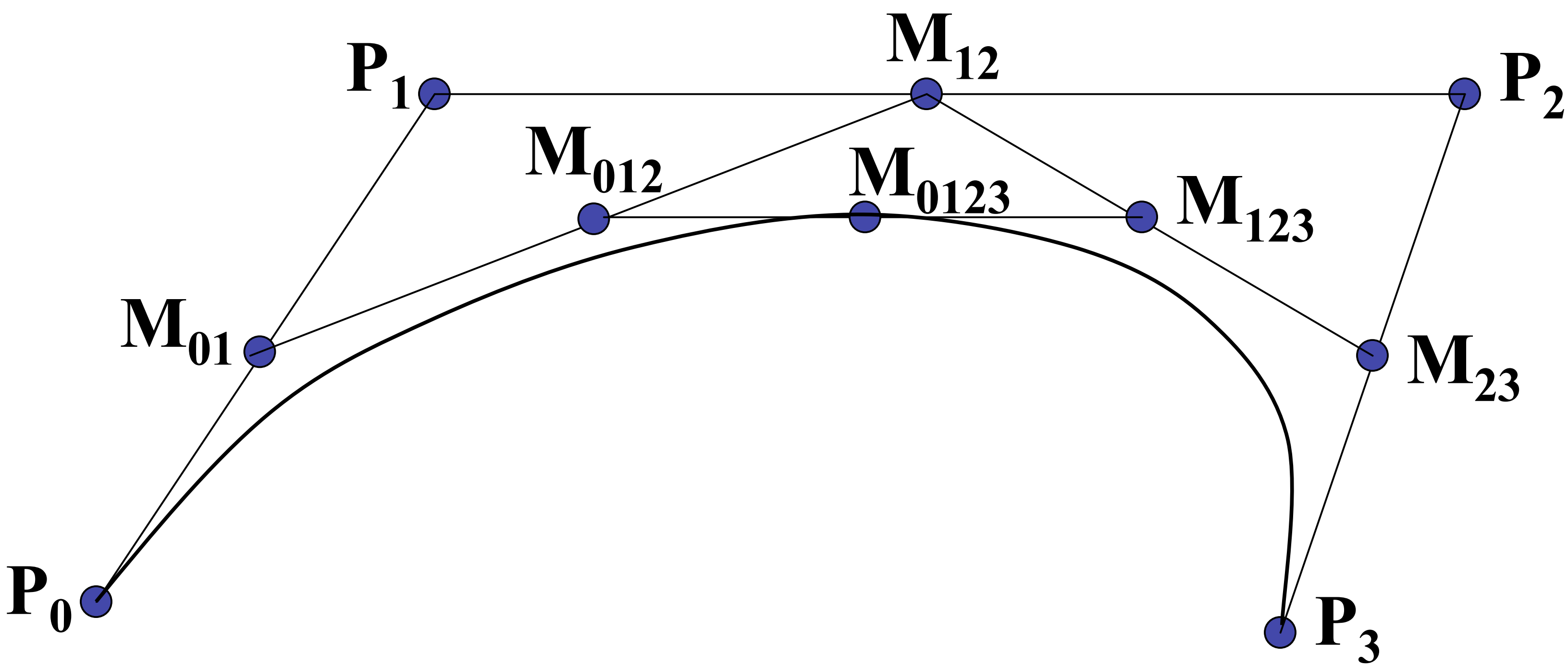
Sub-Dividing Bezier Curves

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



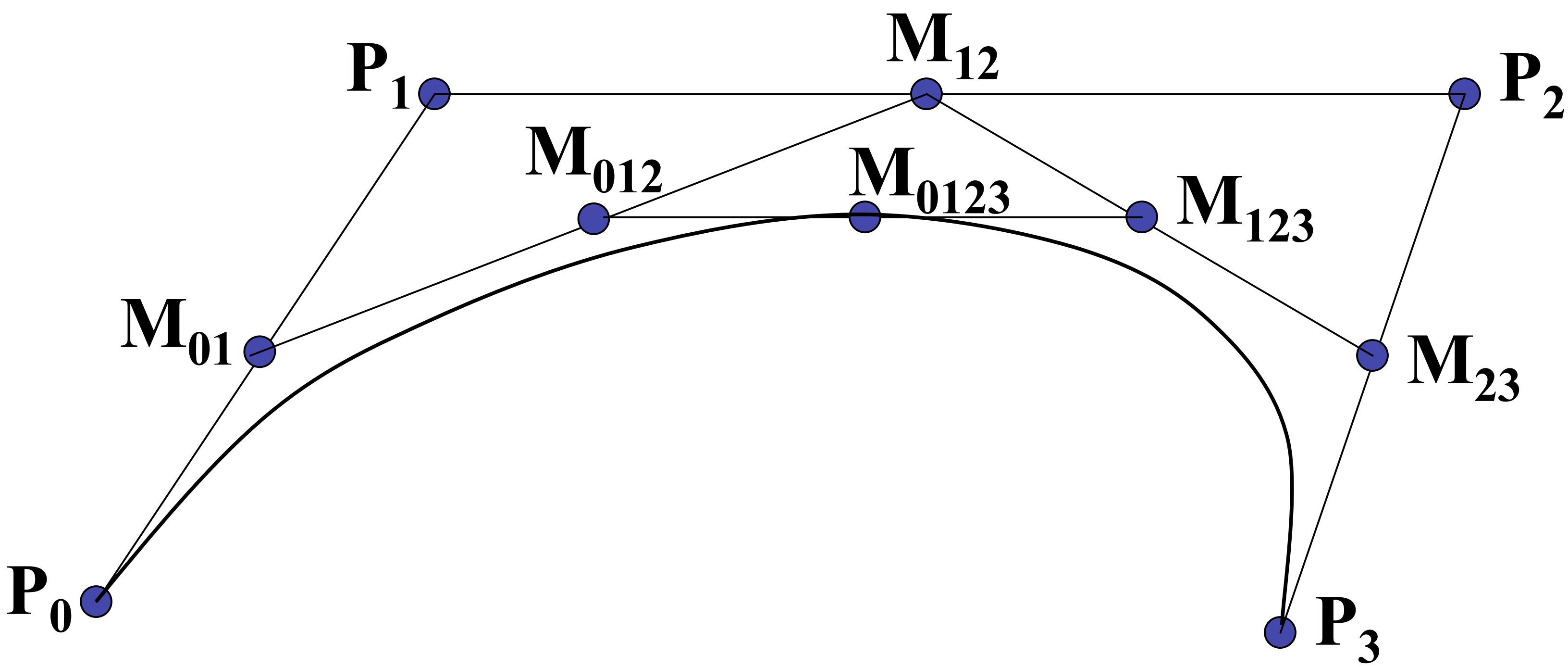
Sub-Dividing Bezier Curves

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



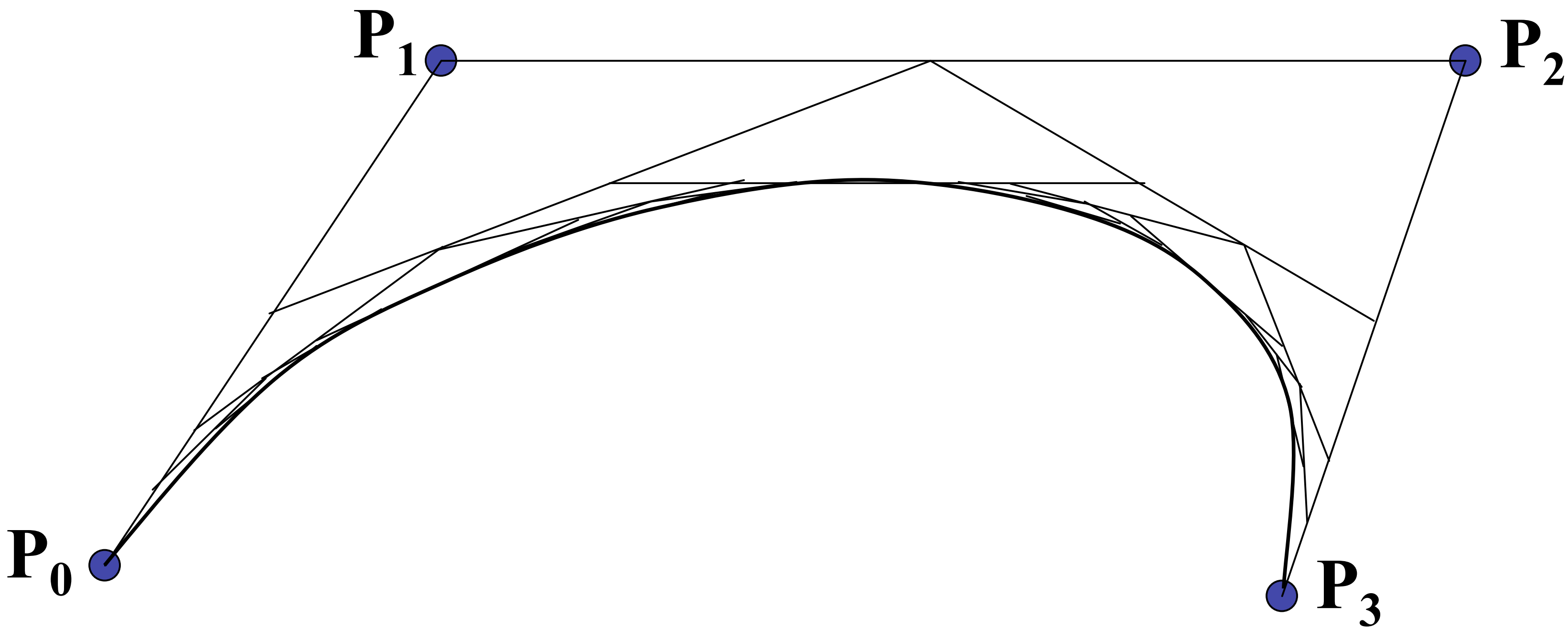
Sub-Dividing Bezier Curves

- 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



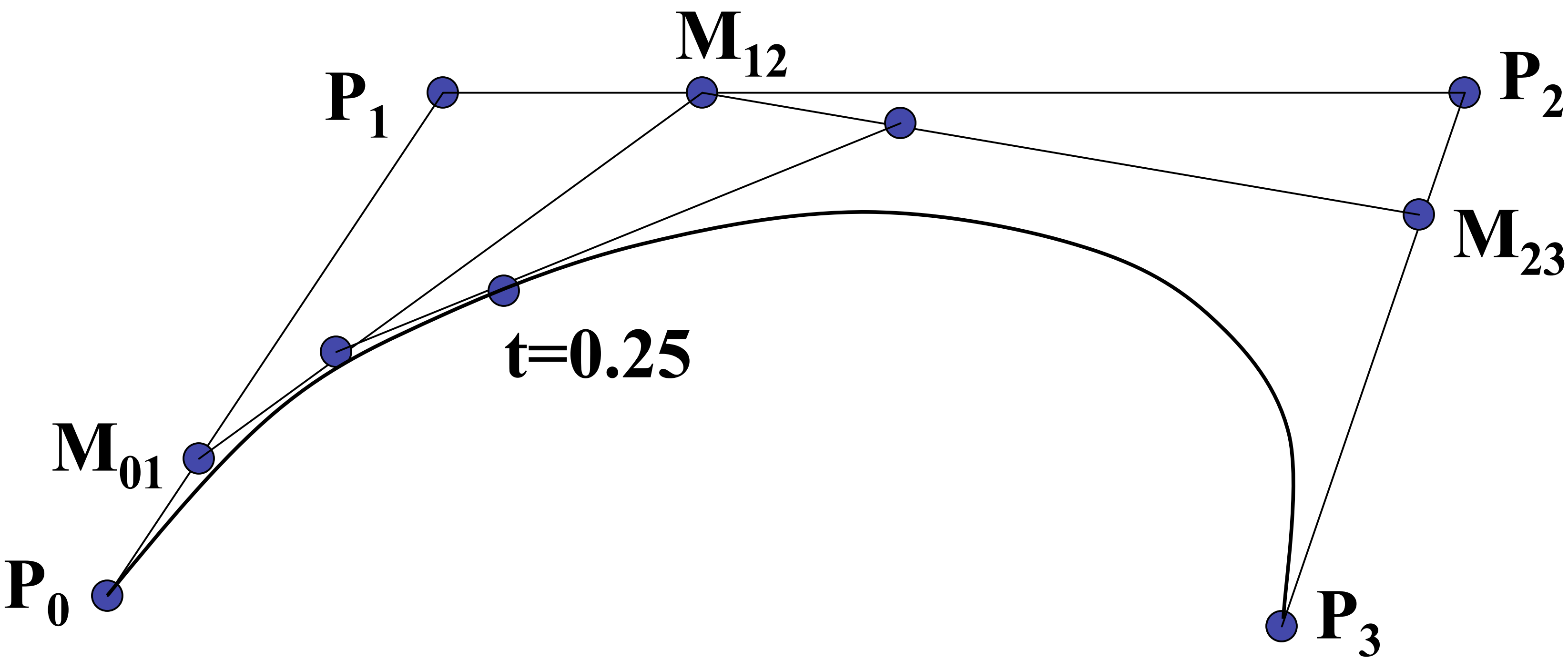
Sub-Dividing Bezier Curves

- 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

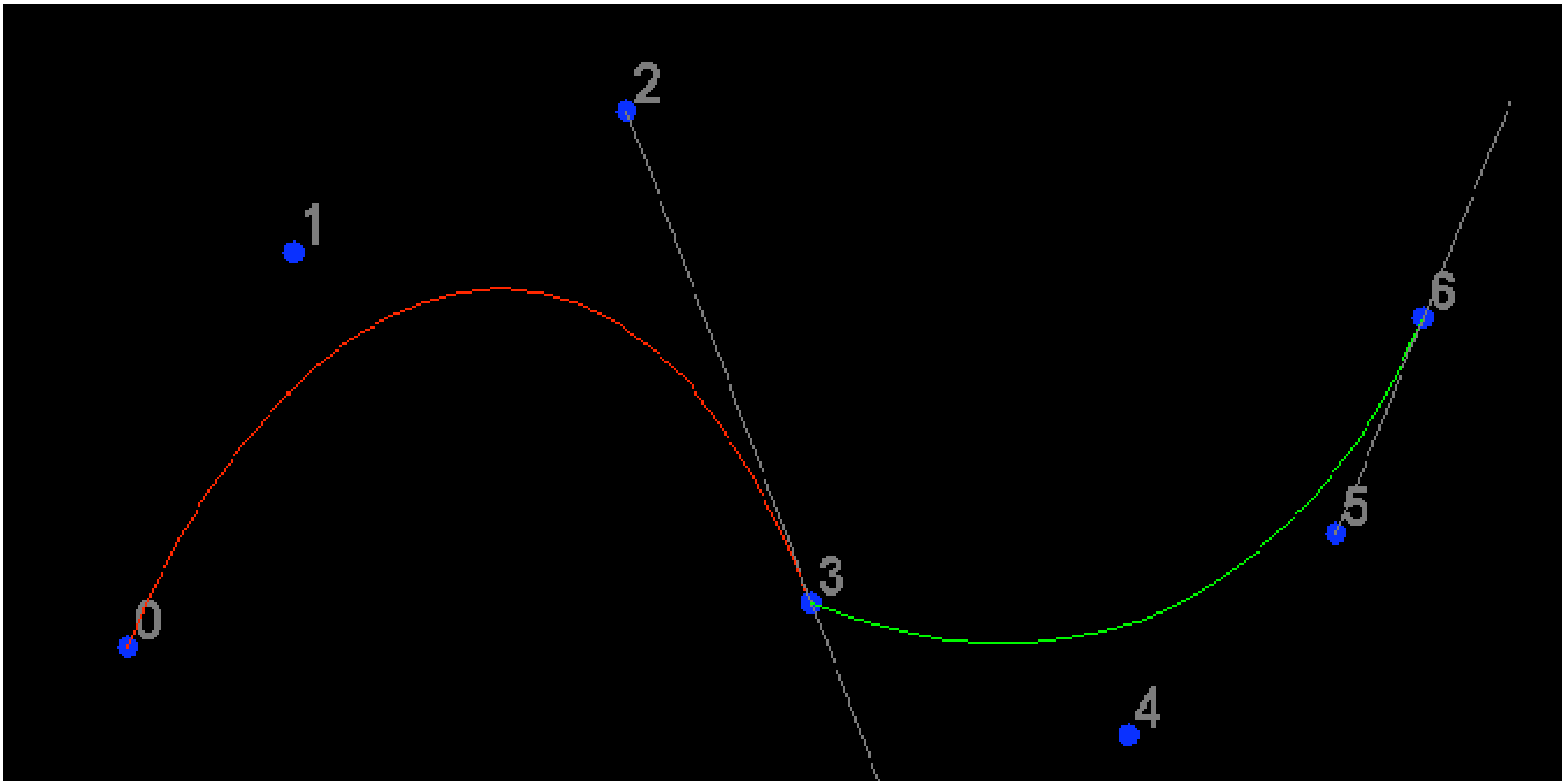


demo: www.saltire.com/applets/advanced_geometry/spline/spline.htm

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

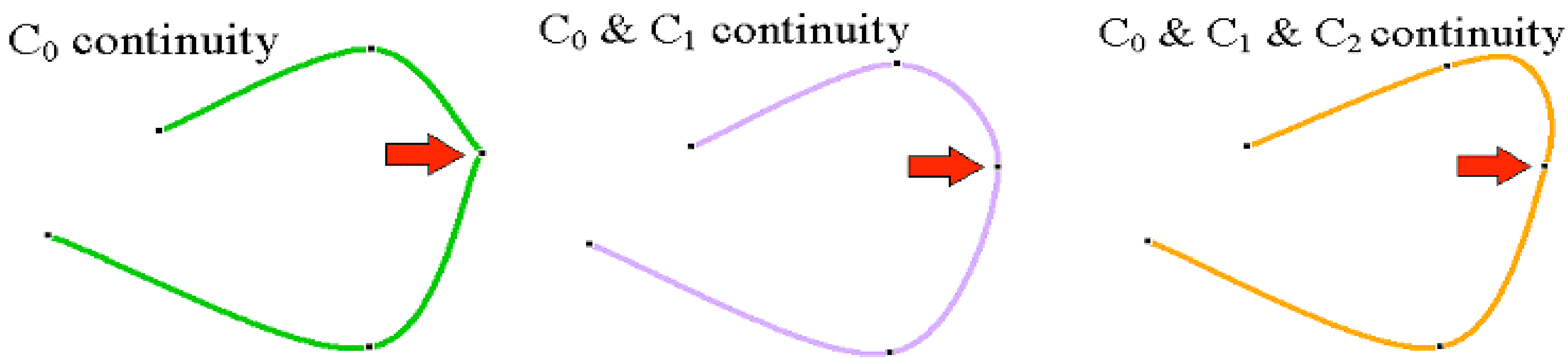
Piecewise Bezier: Continuity Problems



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

Continuity

- when two curves joined, typically want some degree of continuity across knot boundary
  - C0, “C-zero”, point-wise continuous, curves share same point where they join
  - C1, “C-one”, continuous derivatives
  - C2, “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

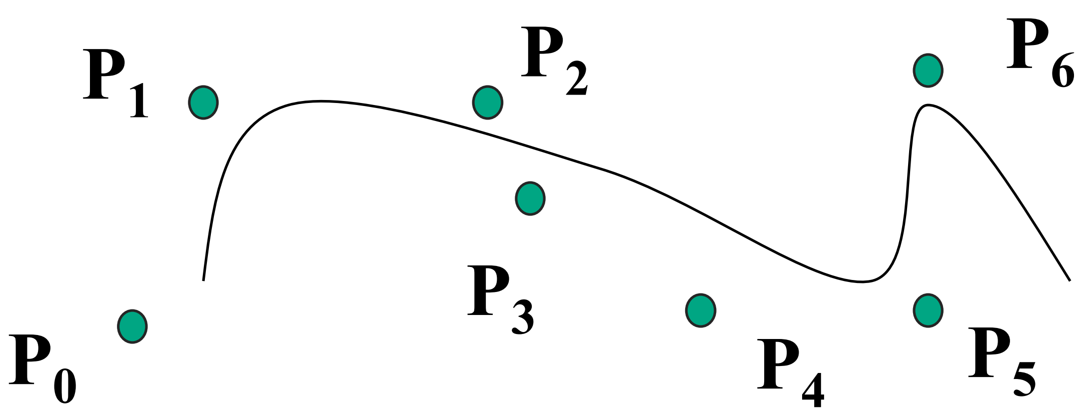
- Hermite curves
  - user specifies derivatives, so C^1 by sharing points and derivatives across knot
- Bezier curves
  - they interpolate endpoints, so C^0 by sharing control pts
  - introduce additional constraints to get C^1
    - parametric derivative is a constant multiple of vector joining first/last 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 $P_{0,1}$ and $P_{1,2}$
  - leads to...

B-Spline Curve

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

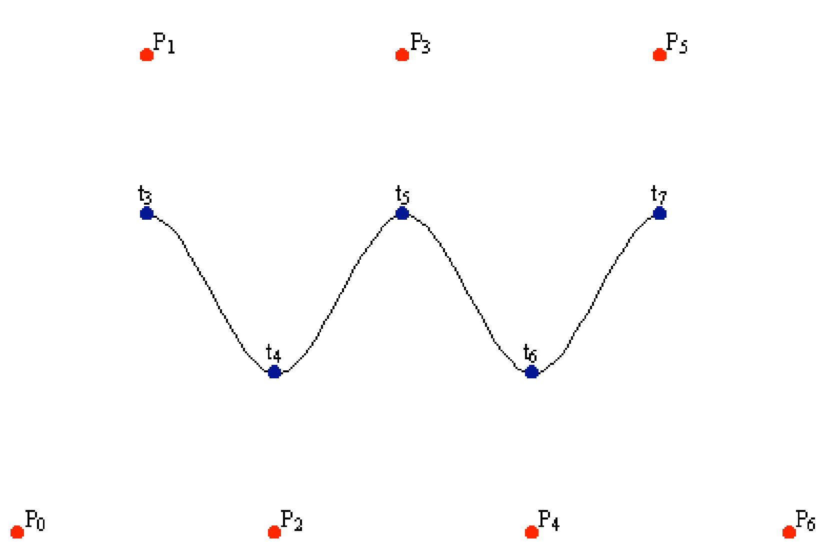
  $(p_{i-2}, p_{i-1}, p_i, p_{i+1})$

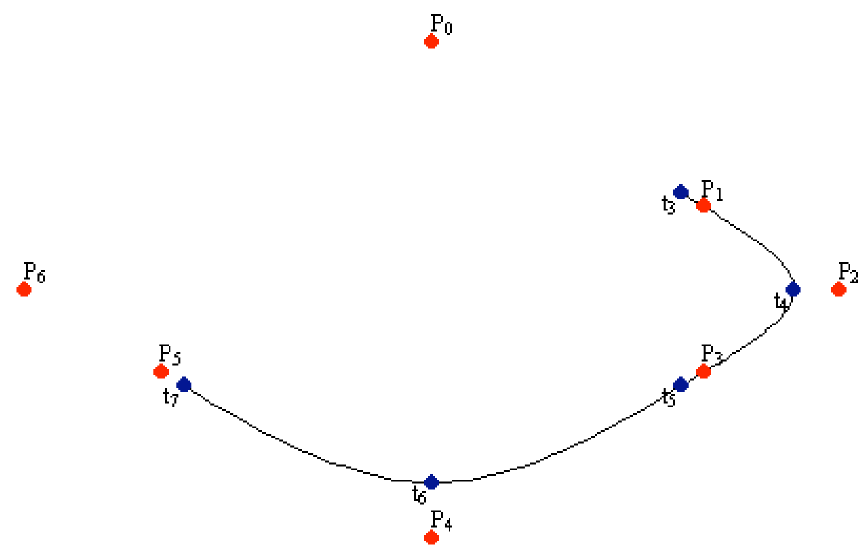
  - Bezier and Hermite goes between p_{i-2} and p_{i+1}
  - B-Spline doesn't interpolate (touch) any of them but approximates the going through p_{i-1} and p_i



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

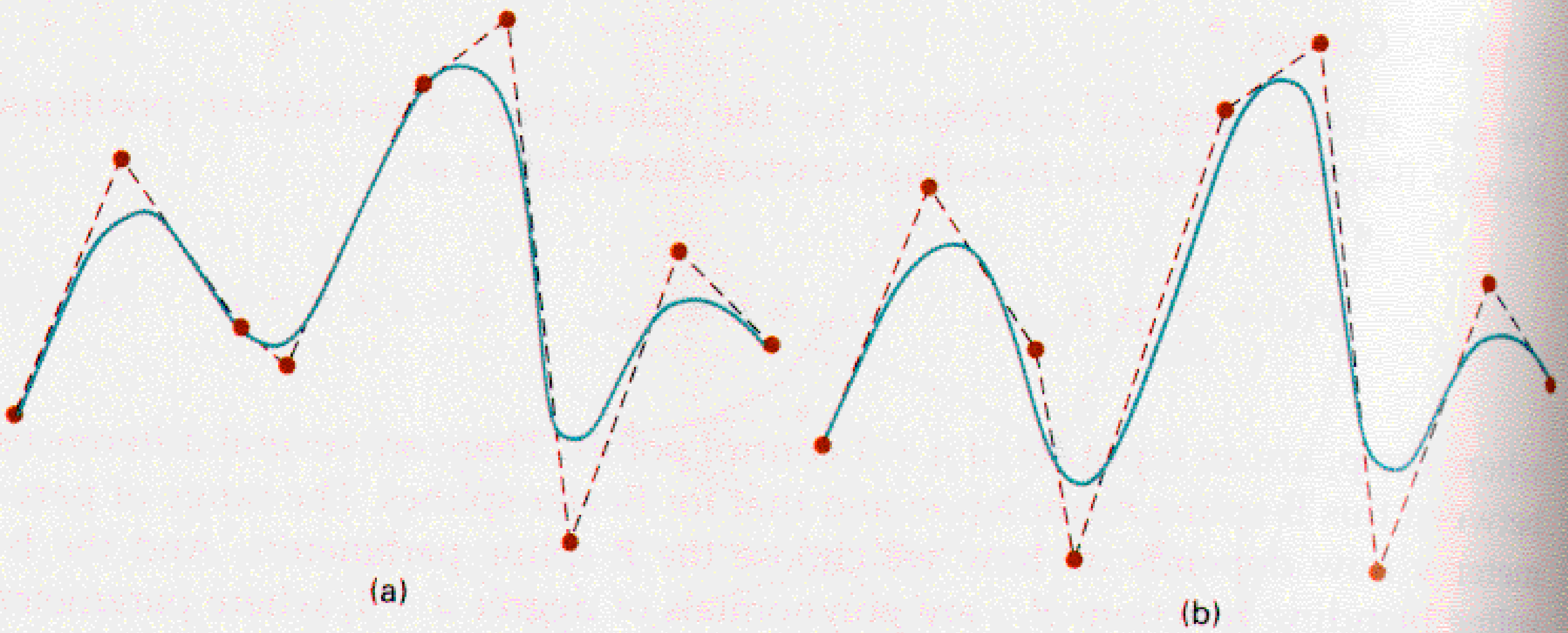


Figure 10-41
Local modification of a B-spline curve. Changing one of the control points in (a) produces curve (b), which is modified only in the neighborhood of the altered control point.