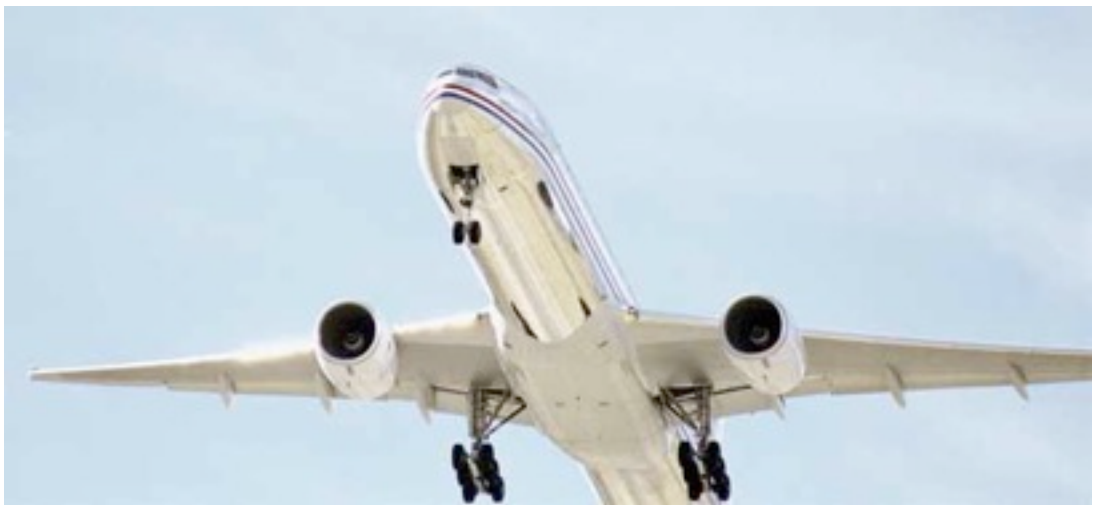
2D Spline Curves

CS 4620 Lecture 18

Motivation: smoothness

- In many applications we need smooth shapes
  - that is, without discontinuities

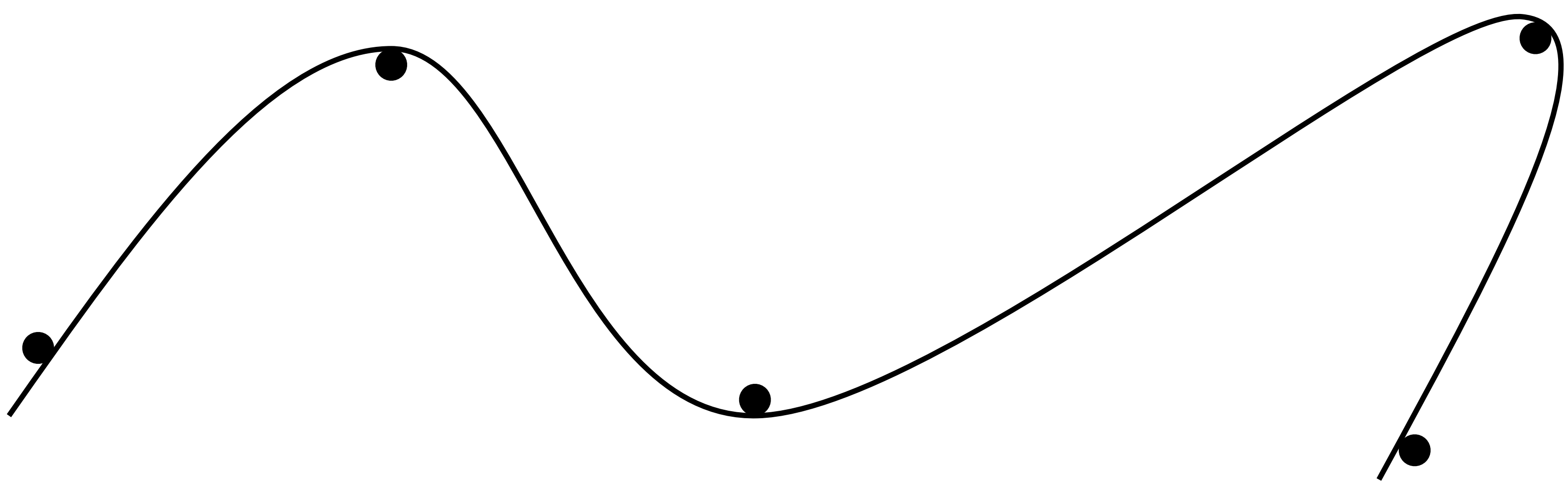


[Boeing]

- So far we can make
  - things with corners (lines, triangles, squares, rectangles, …)
  - circles, ellipses, other special shapes (only get you so far!)

Classical approach

- Pencil-and-paper draftsmen also needed smooth curves
- Origin of "spline:" strip of flexible metal
  - held in place by pegs or weights to constrain shape
  - traced to produce smooth contour



Translating into usable math

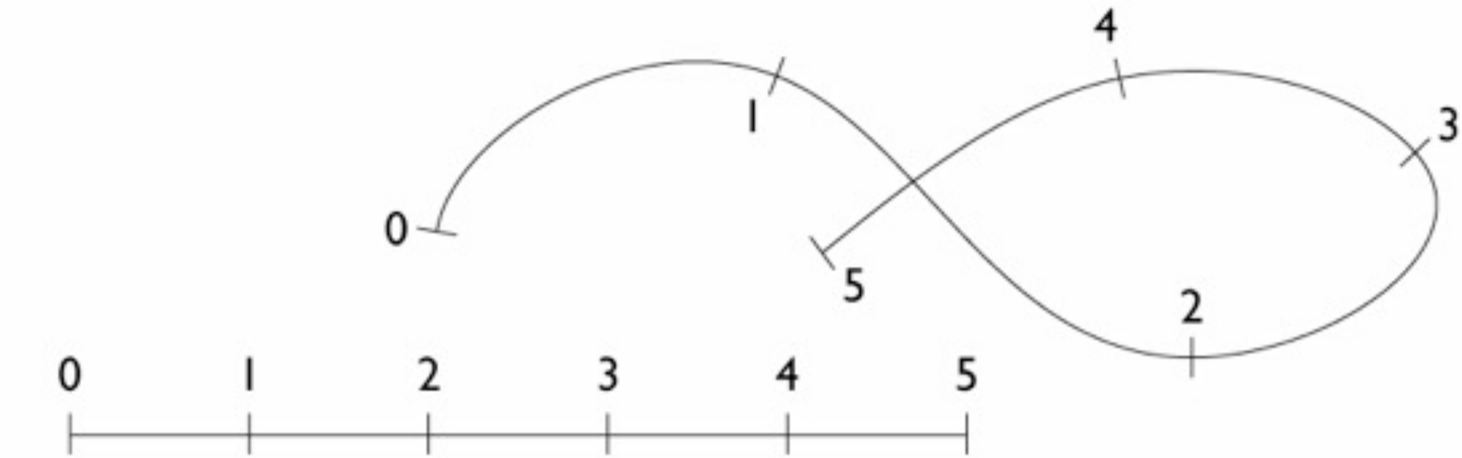
- Smoothness
  - in drafting spline, comes from physical curvature minimization
  - in CG spline, comes from choosing smooth functions
    - usually low-order polynomials
- Control
  - in drafting spline, comes from fixed pegs
  - in CG spline, comes from user-specified *control points*

Defining spline curves

- At the most general they are parametric curves

$$S = \{\mathbf{f}(t) \mid t \in [0, N]\}$$

- For splines, **f**(*t*) is piecewise polynomial
  - for this lecture, the discontinuities are at the integers

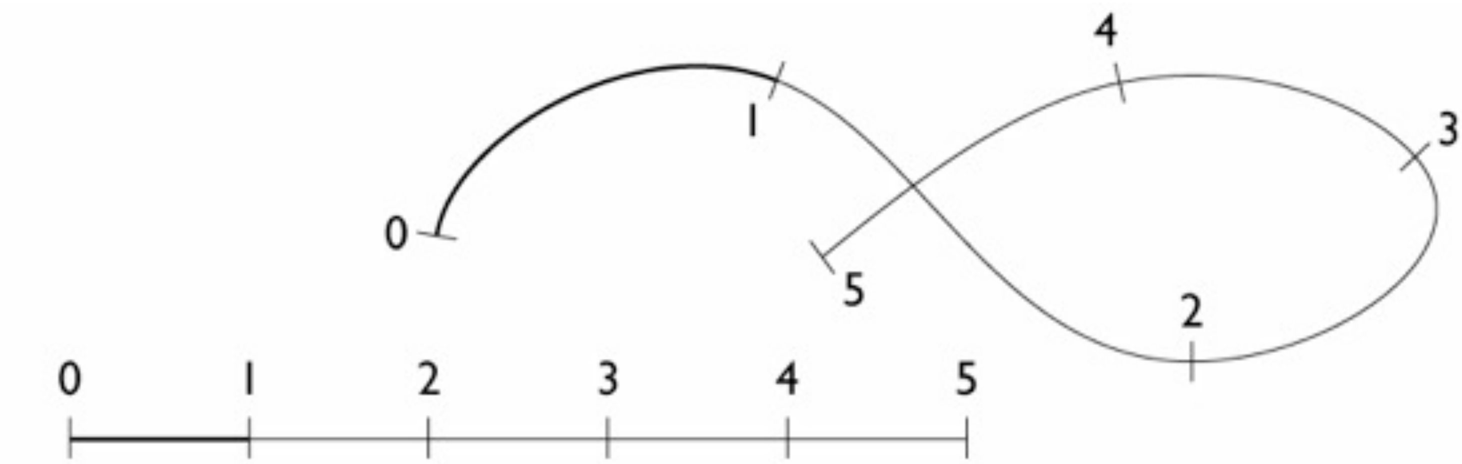


Defining spline curves

- At the most general they are parametric curves

$$S = \{\mathbf{f}(t) \mid t \in [0, N]\}$$

- For splines, **f**(t) is piecewise polynomial
  - for this lecture, the discontinuities are at the integers

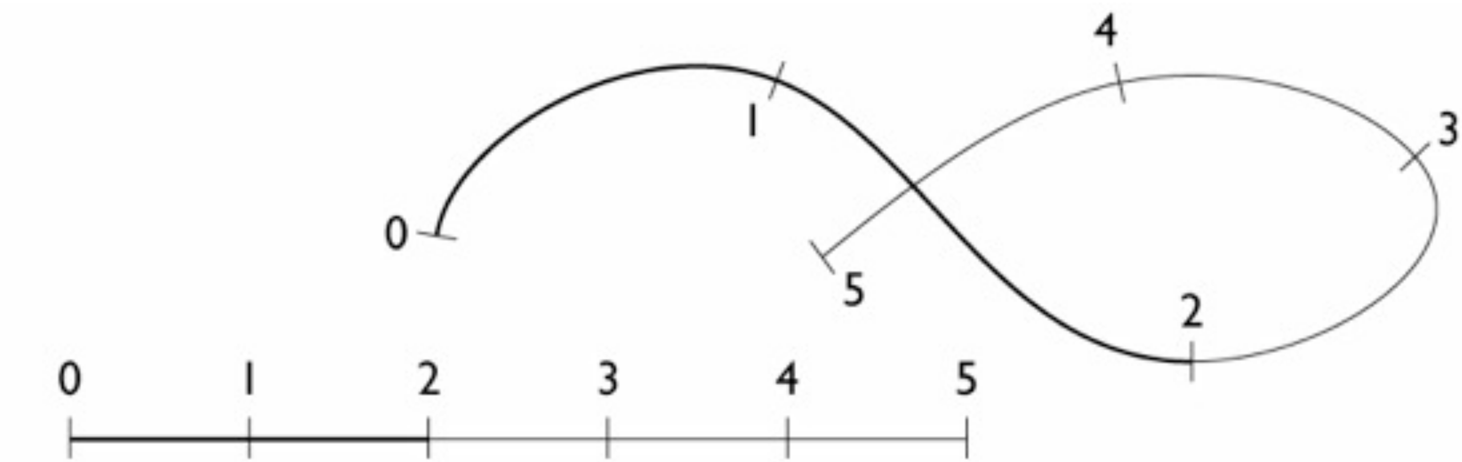


Defining spline curves

- At the most general they are parametric curves

$$S = \{\mathbf{f}(t) \mid t \in [0, N]\}$$

- For splines, **f**(*t*) is piecewise polynomial
  - for this lecture, the discontinuities are at the integers

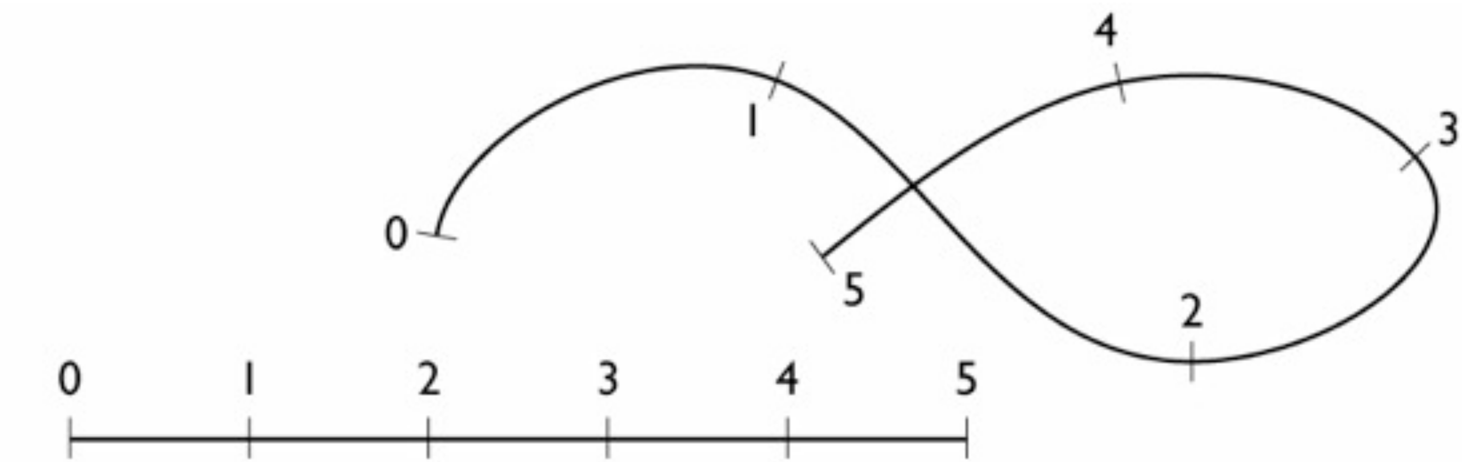


Defining spline curves

- At the most general they are parametric curves

$$S = \{\mathbf{f}(t) \,|\, t \in [0, N]\}$$

- For splines, **f**(*t*) is piecewise polynomial
  - for this lecture, the discontinuities are at the integers



Defining spline curves

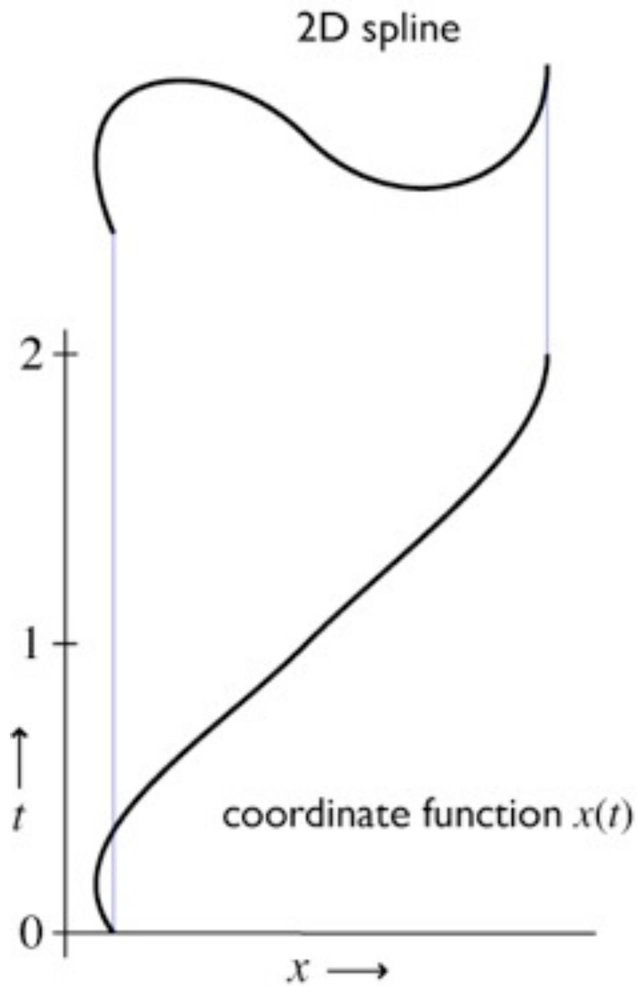
- Generally **f**(*t*) is a piecewise polynomial
  - for this lecture, the discontinuities are at the integers
  - e.g., a cubic spline has the following form over [k, k + 1]:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

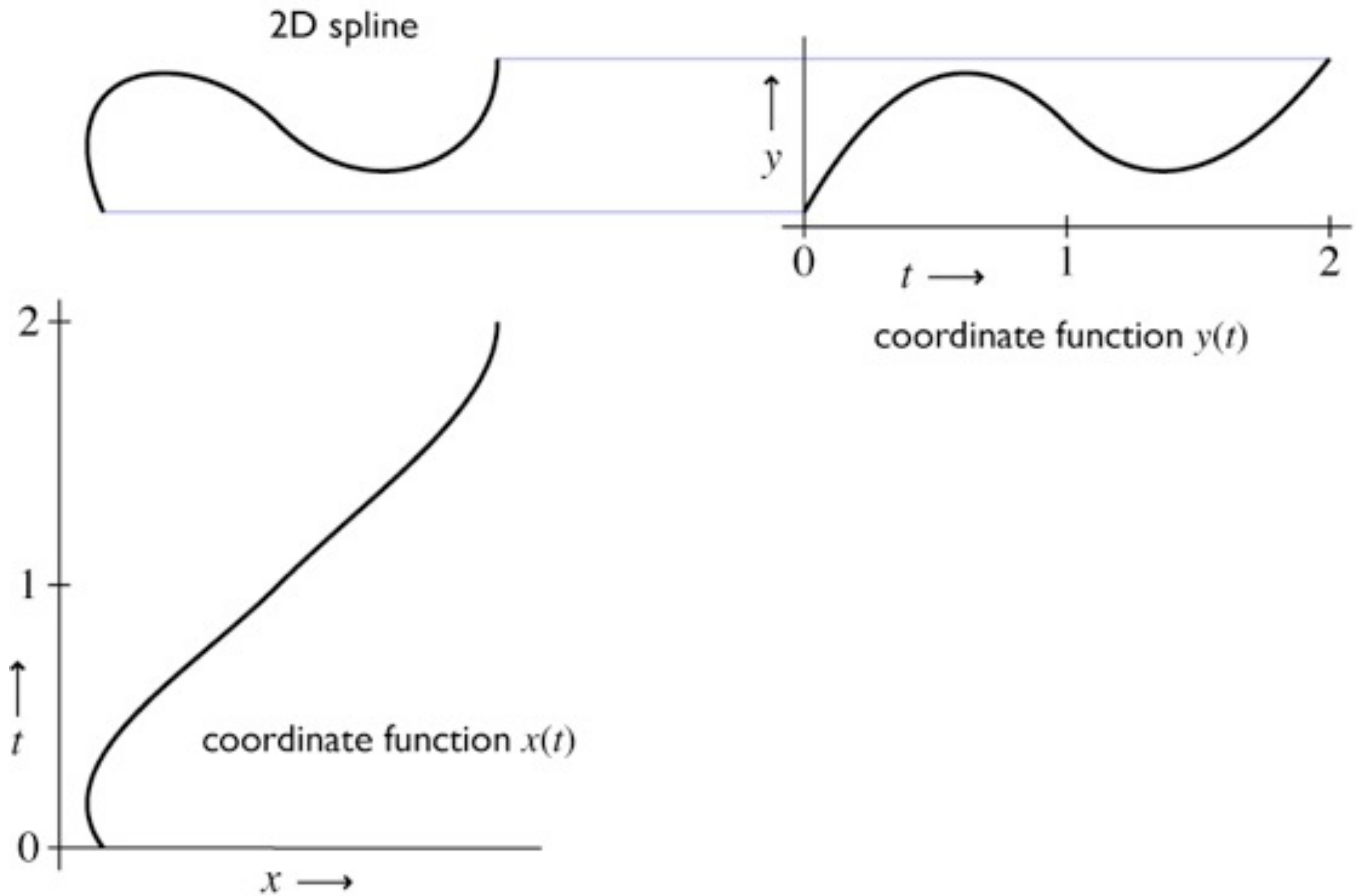
  - Coefficients are different for every interval

Coordinate functions

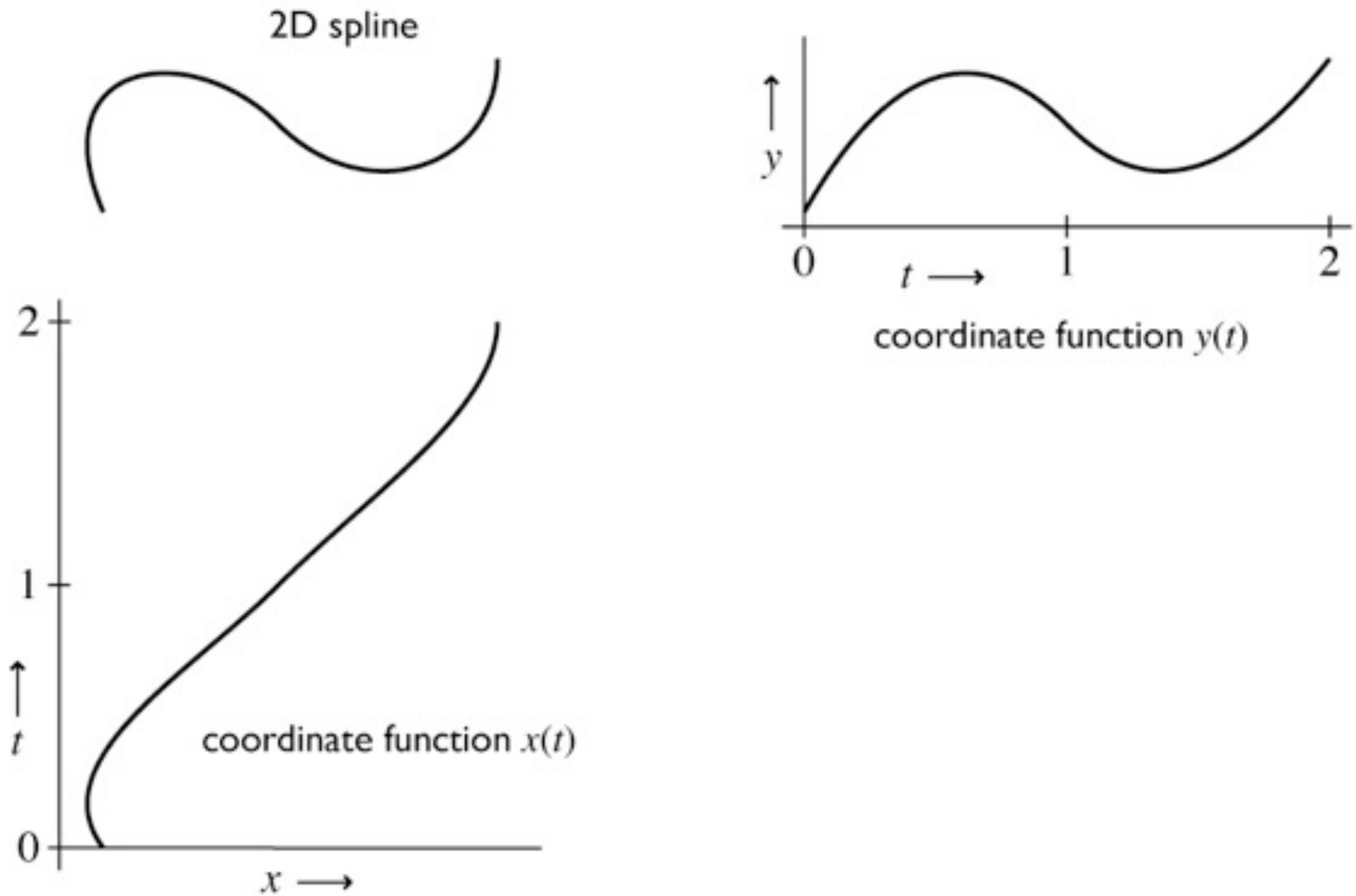
Coordinate functions



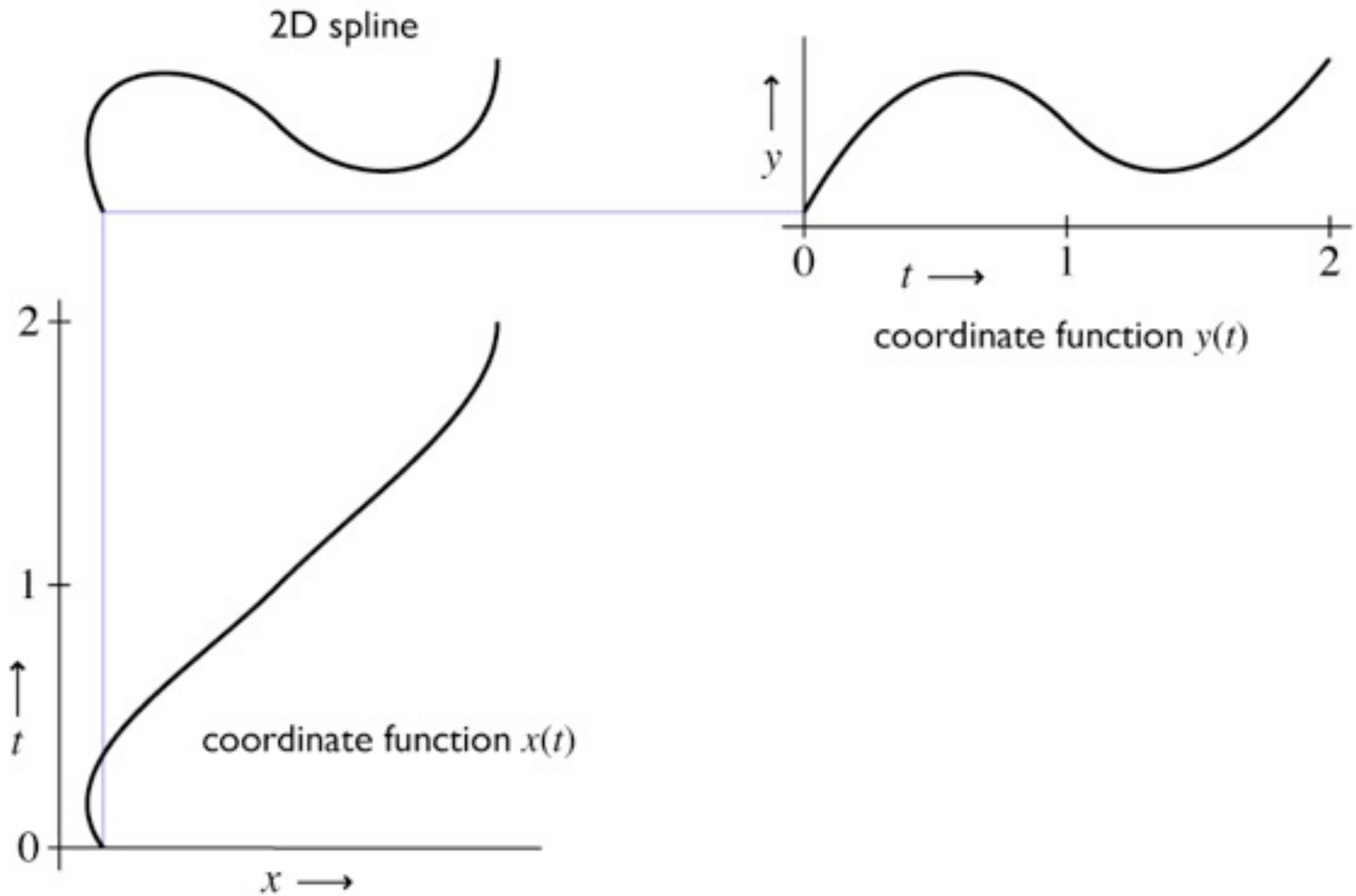
Coordinate functions



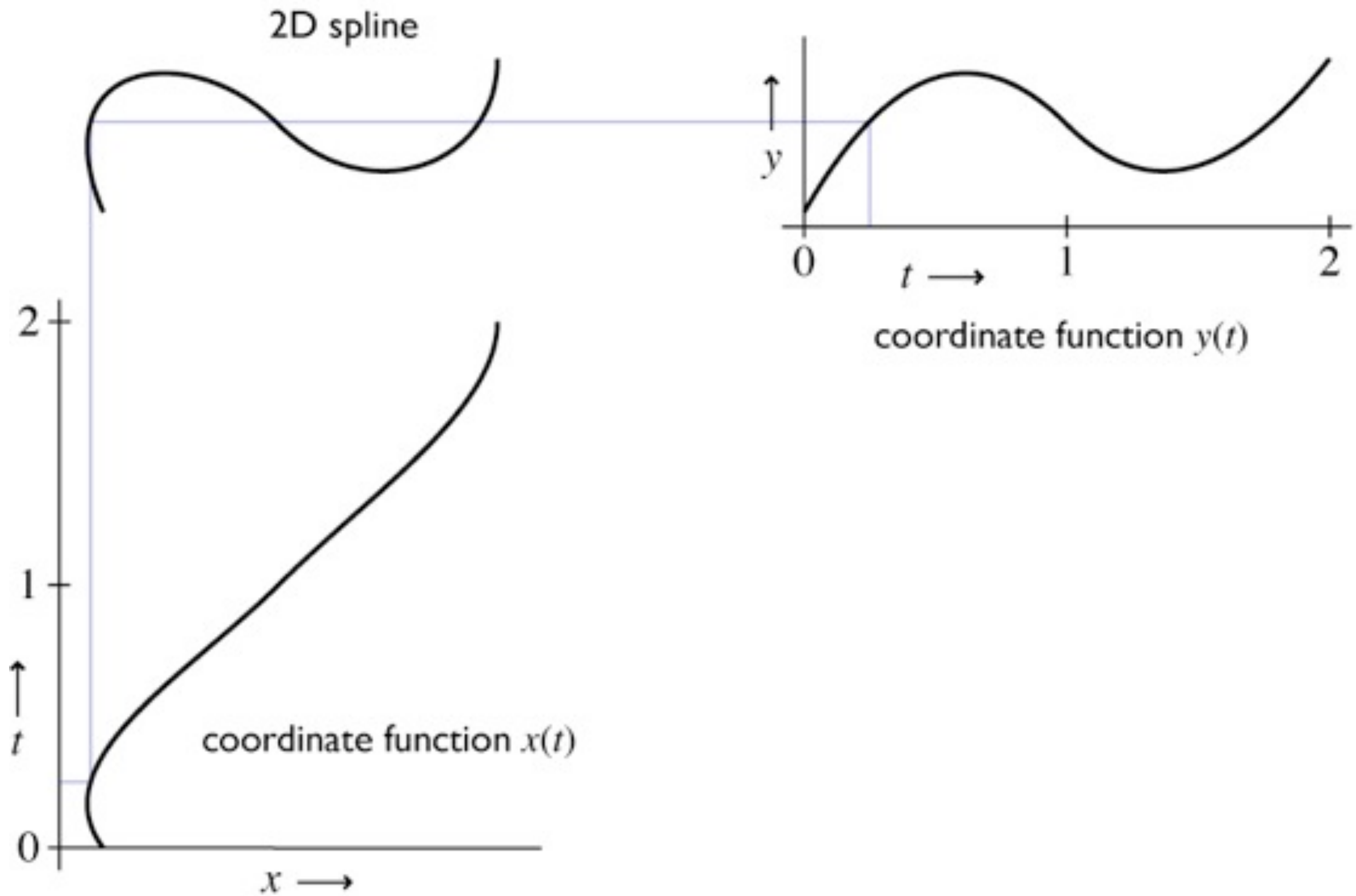
Coordinate functions



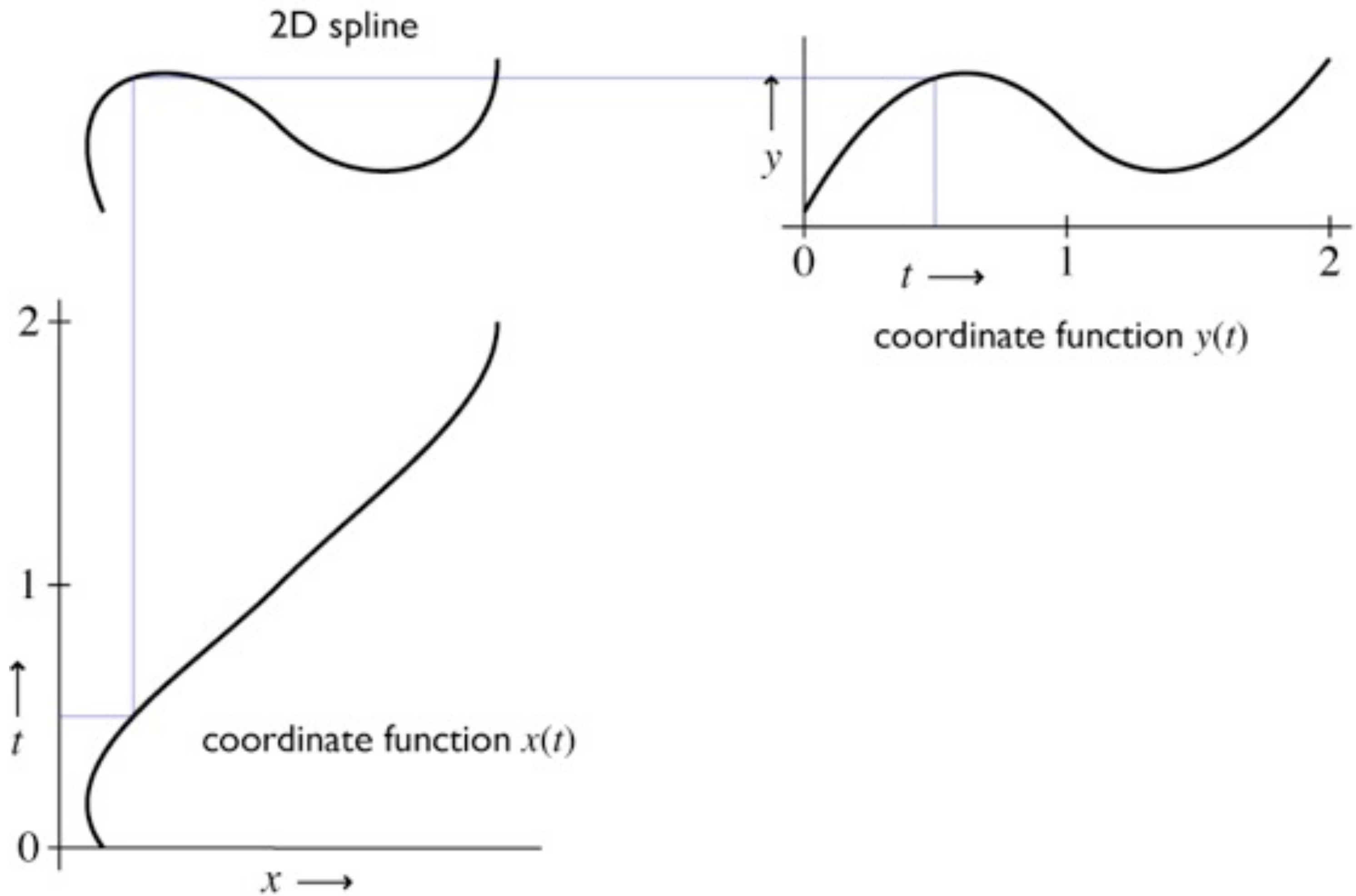
Coordinate functions



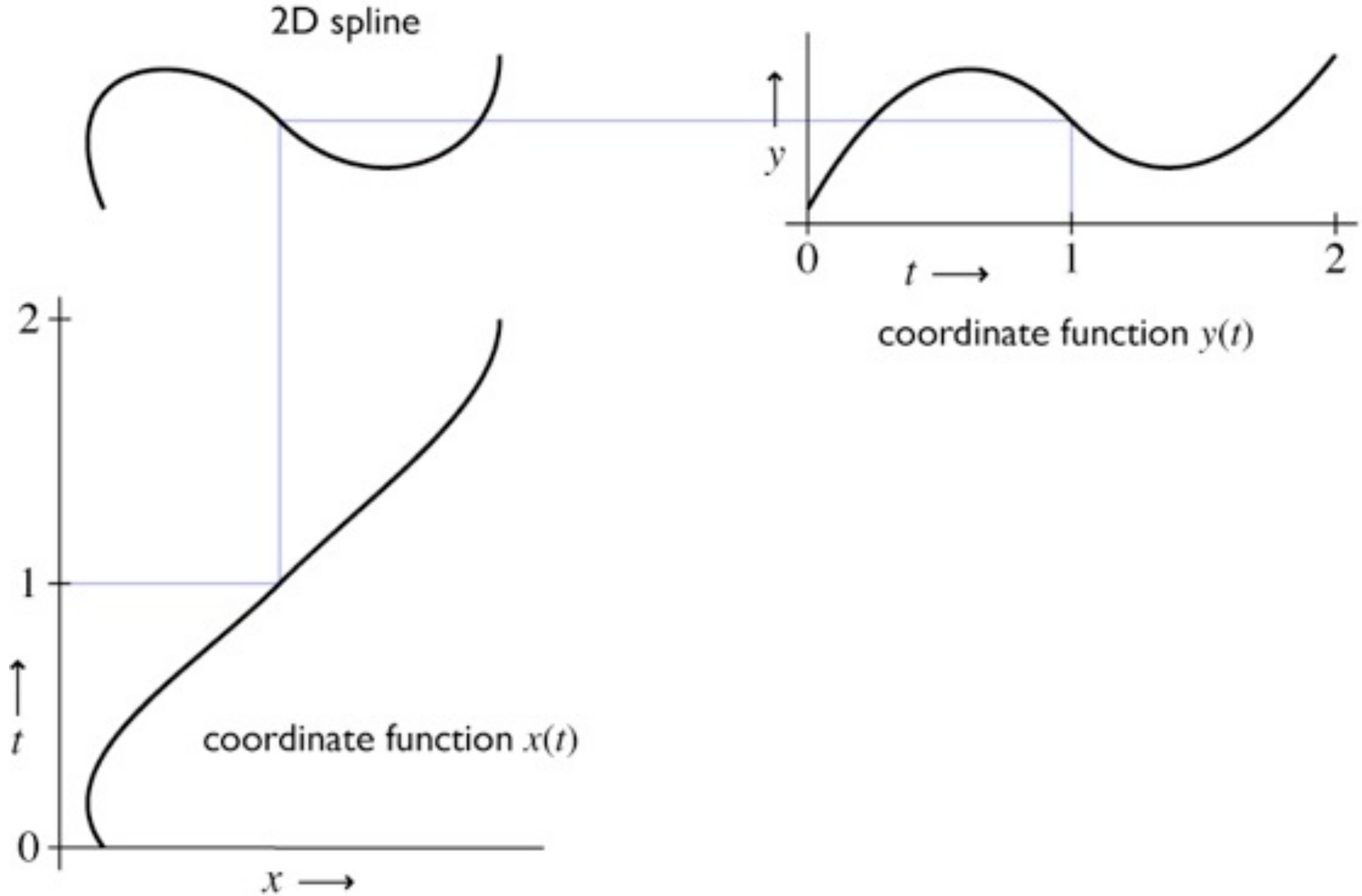
Coordinate functions



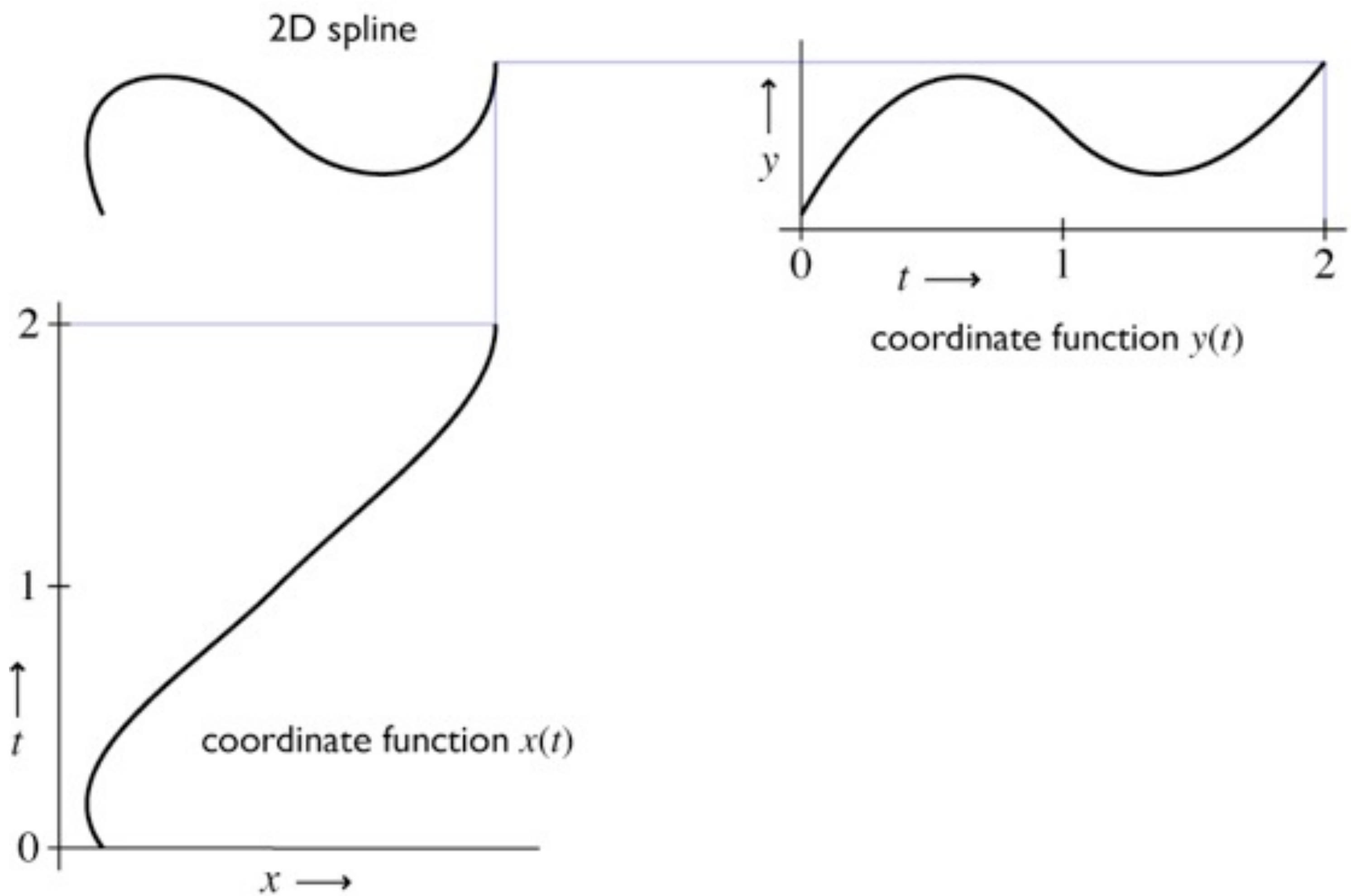
Coordinate functions



Coordinate functions

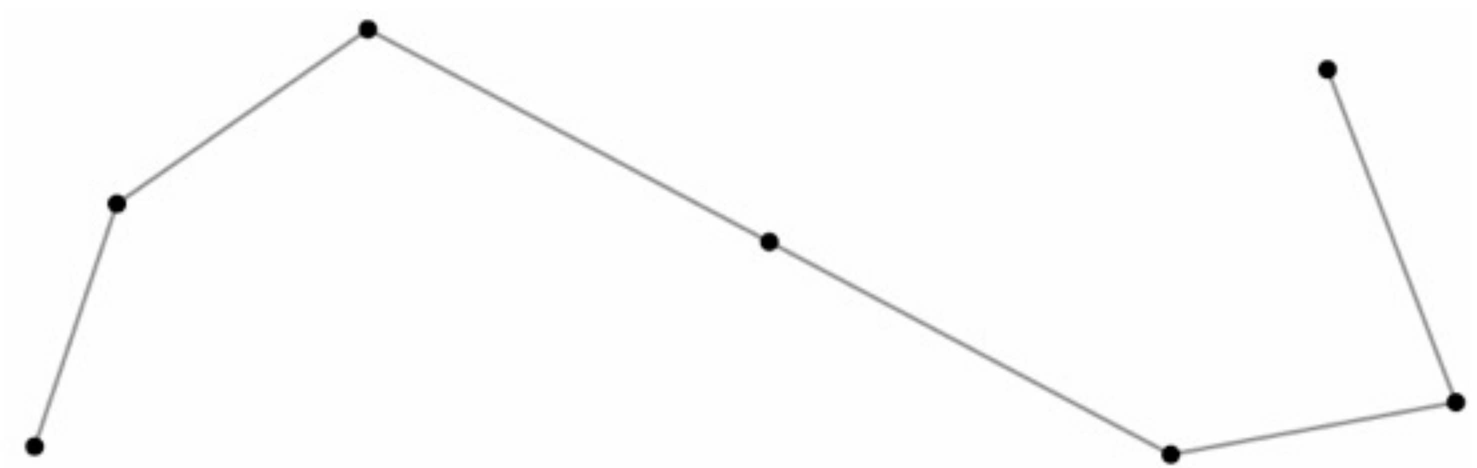


Coordinate functions



Control of spline curves

- Specified by a sequence of controls (points or vectors)
- Shape is guided by control points (aka control polygon)
  - interpolating: passes through points
  - approximating: merely guided by points

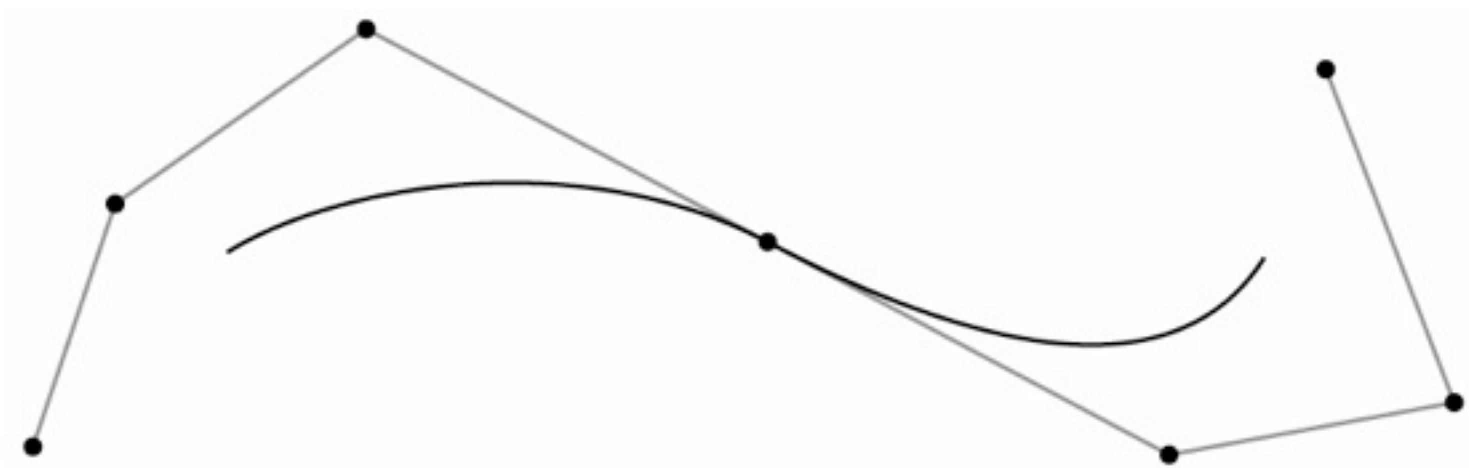


Control of spline curves

- Specified by a sequence of controls (points or vectors)
- Shape is guided by control points (aka control polygon)
  - interpolating: passes through points
  - approximating: merely guided by points

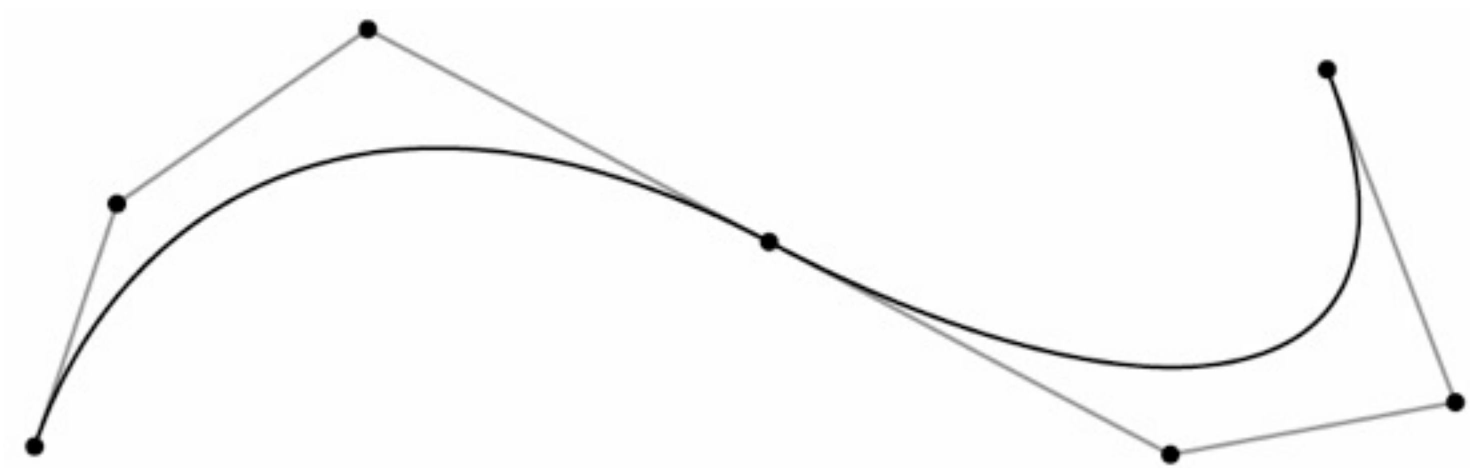
Control of spline curves

- Specified by a sequence of controls (points or vectors)
- Shape is guided by control points (aka control polygon)
  - interpolating: passes through points
  - approximating: merely guided by points



Control of spline curves

- Specified by a sequence of controls (points or vectors)
- Shape is guided by control points (aka control polygon)
  - interpolating: passes through points
  - approximating: merely guided by points



How splines depend on their controls

- Each coordinate is separate
  - the function $x(t)$ is determined solely by the x coordinates of the control points
  - this means 1D, 2D, 3D, … curves are all really the same
- Spline curves are **linear** functions of their controls
  - moving a control point two inches to the right moves $x(t)$ twice as far as moving it by one inch
  - $x(t)$, for fixed t, is a linear combination (weighted sum) of the controls' x coordinates
  - $\mathbf{f}(t)$, for fixed t, is a linear combination (weighted sum) of the controls

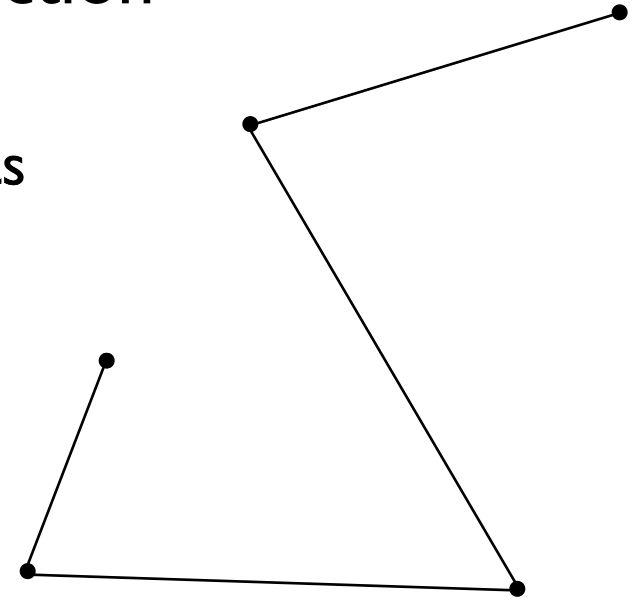
Plan

1. Spline segments
   - how to define a polynomial on [0,1]
   - …that has the properties you want
   - …and is easy to control
2. Spline curves
   - how to chain together lots of segments
   - …so that the whole curve has the properties you want
   - …and is easy to control
3. Refinement and evaluation
   - how to add detail to splines
   - how to approximate them with line segments

Spline Segments

Trivial example: piecewise linear

- This spline is just a polygon
  - control points are the vertices
- But we can derive it anyway as an illustration
- Each interval will be a linear function
  - $x(t) = at + b$
  - constraints are values at endpoints
  - $b = x_0$; $a = x_1 - x_0$
  - this is linear interpolation



Trivial example: piecewise linear

- Vector formulation

$$x(t) = (x_1 - x_0)t + x_0$$
$$y(t) = (y_1 - y_0)t + y_0$$
$$\mathbf{f}(t) = (\mathbf{p}_1 - \mathbf{p}_0)t + \mathbf{p}_0$$

- Matrix formulation

$$\mathbf{f}(t) = \begin{bmatrix} t & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \end{bmatrix}$$

Trivial example: piecewise linear

- Basis function formulation
  - regroup expression by **p** rather than *t*

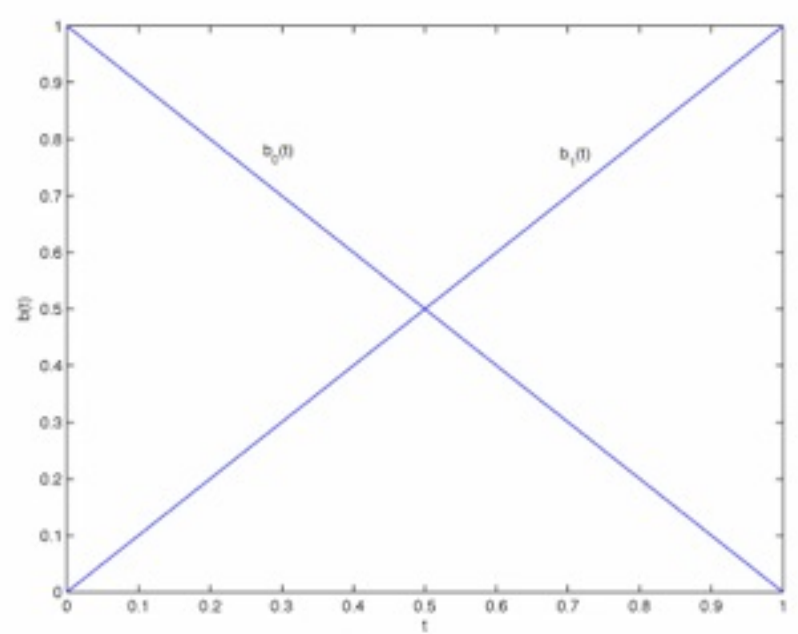
$$\begin{aligned}\mathbf{f}(t) &= (\mathbf{p}_1 - \mathbf{p}_0)t + \mathbf{p}_0 \\ &= (1-t)\mathbf{p}_0 + t\mathbf{p}_1\end{aligned}$$

  - interpretation in matrix viewpoint

$$\mathbf{f}(t) = \left(\begin{bmatrix} t & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \right) \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \end{bmatrix}$$

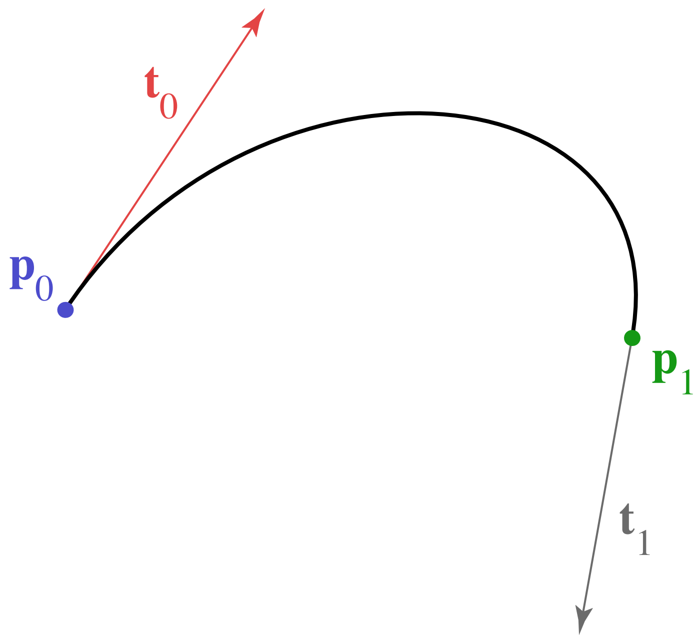
Trivial example: piecewise linear

- Vector blending formulation: "average of points"
  - blending functions: contribution of each point as *t* changes



Hermite splines

- Less trivial example
- Form of curve: piecewise cubic
- Constraints: endpoints and tangents (derivatives)



Hermite splines

- Solve constraints to find coefficients

$$x(t) = at^3 + bt^2 + ct + d$$
$$x'(t) = 3at^2 + 2bt + c$$
$$x(0) = x_0 = d$$
$$x(1) = x_1 = a + b + c + d$$
$$x'(0) = x'_0 = c$$
$$x'(1) = x'_1 = 3a + 2b + c$$

$$d = x_0$$
$$c = x'_0$$
$$a = 2x_0 - 2x_1 + x'_0 + x'_1$$
$$b = -3x_0 + 3x_1 - 2x'_0 - x'_1$$

Matrix form of spline

$$\mathbf{f}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

$$\mathbf{f}(t) = b_0(t)\mathbf{p}_0 + b_1(t)\mathbf{p}_1 + b_2(t)\mathbf{p}_2 + b_3(t)\mathbf{p}_3$$

Matrix form of spline

$$\mathbf{f}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

$$\mathbf{f}(t) = b_0(t)\mathbf{p}_0 + b_1(t)\mathbf{p}_1 + b_2(t)\mathbf{p}_2 + b_3(t)\mathbf{p}_3$$

Matrix form of spline

$$\mathbf{f}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

$$\mathbf{f}(t) = b_0(t)\mathbf{p}_0 + b_1(t)\mathbf{p}_1 + b_2(t)\mathbf{p}_2 + b_3(t)\mathbf{p}_3$$

Hermite splines

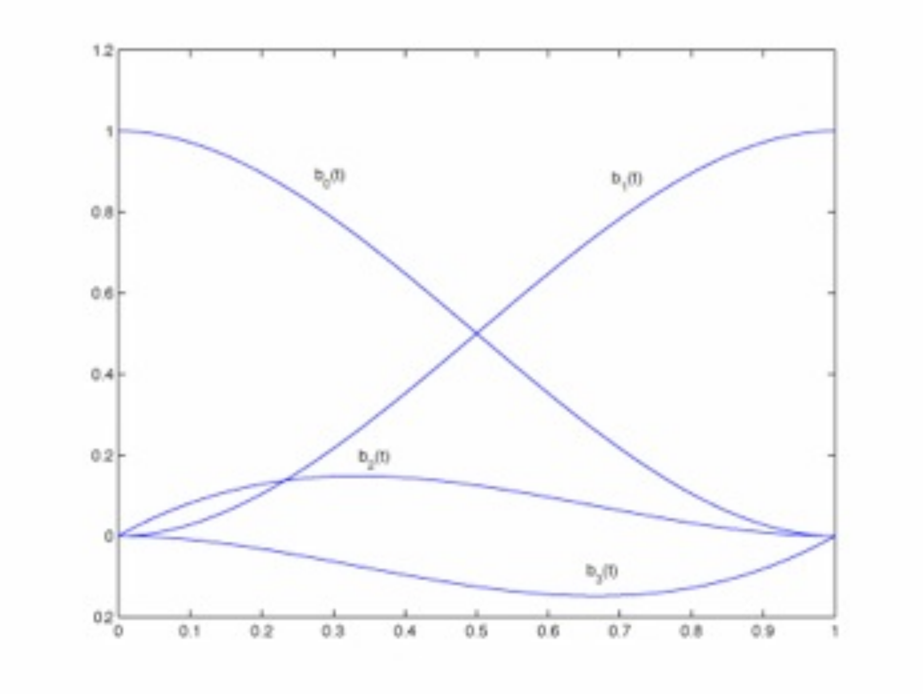
- Matrix form is much simpler

$$\mathbf{f}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{t}_0 \\ \mathbf{t}_1 \end{bmatrix}$$

  – coefficients = rows
  – basis functions = columns
    - note **p** columns sum to $[0\ 0\ 0\ 1]^T$

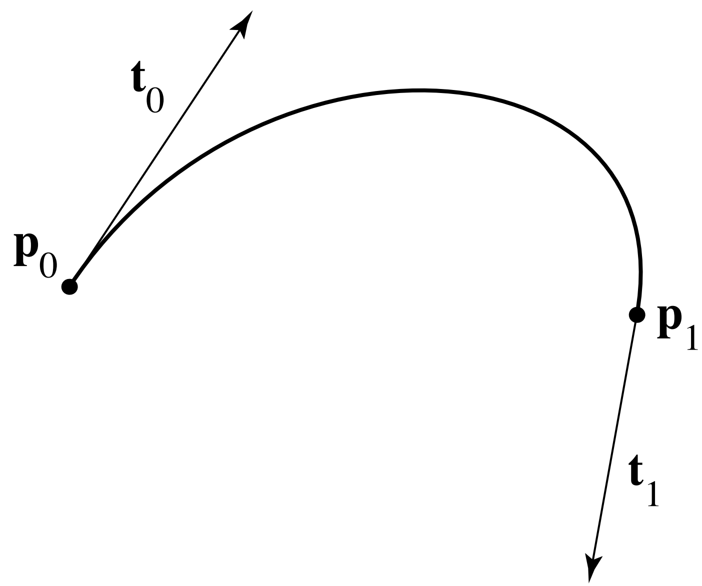
Hermite splines

- Hermite blending functions



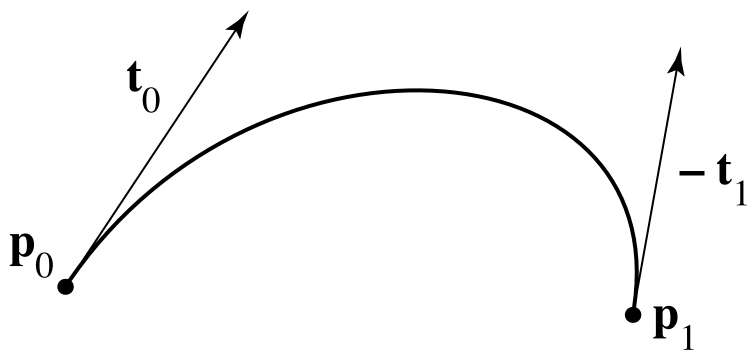
Hermite to Bézier

- Mixture of points and vectors is awkward
- Specify tangents as differences of points



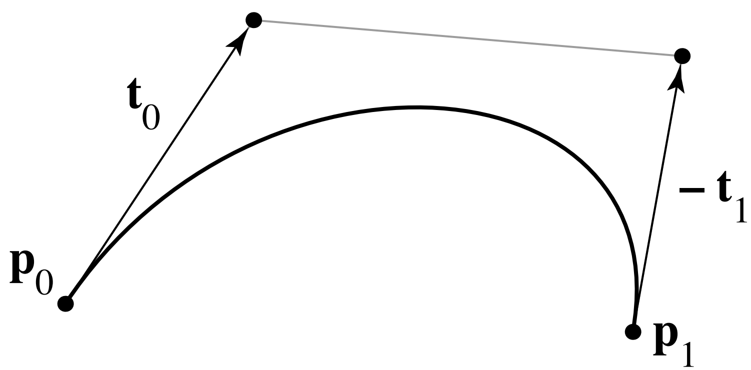
Hermite to Bézier

- Mixture of points and vectors is awkward
- Specify tangents as differences of points



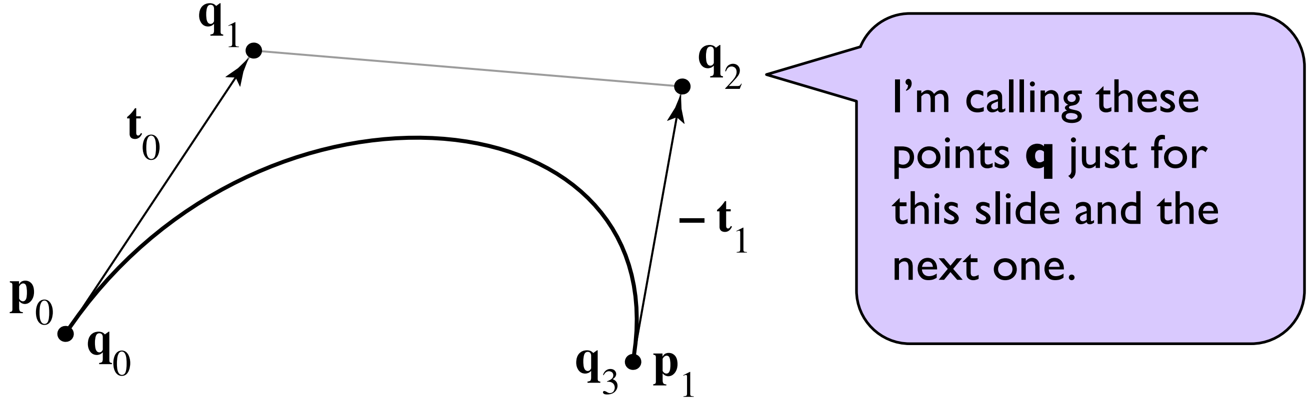
Hermite to Bézier

- Mixture of points and vectors is awkward
- Specify tangents as differences of points



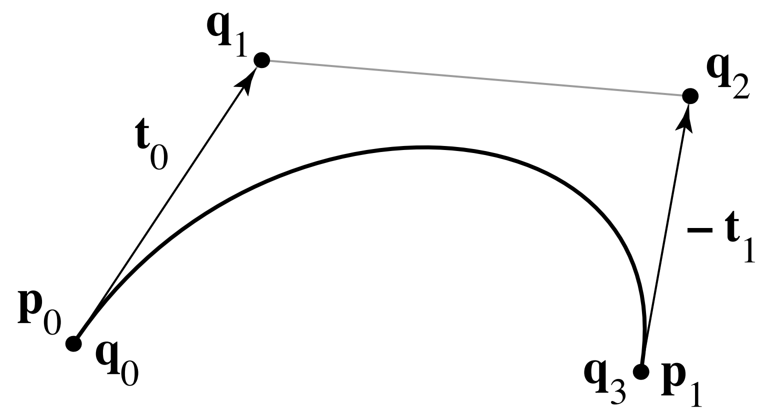
Hermite to Bézier

- Mixture of points and vectors is awkward
- Specify tangents as differences of points



  – note derivative is defined as 3 times offset
    - reason is illustrated by linear case

Hermite to Bézier



$$\mathbf{p}_0 = \mathbf{q}_0$$

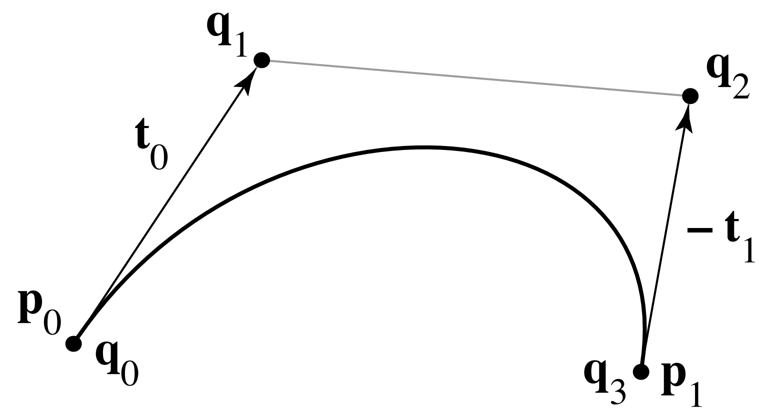
$$\mathbf{p}_1 = \mathbf{q}_3$$

$$\mathbf{t}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$

$$\mathbf{t}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$

$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}$$

Hermite to Bézier



$$\mathbf{p}_0 = \mathbf{q}_0$$

$$\mathbf{p}_1 = \mathbf{q}_3$$

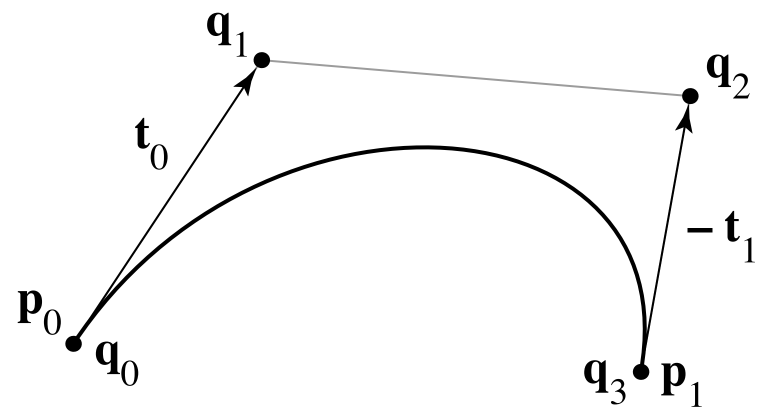
$$\mathbf{t}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$

$$\mathbf{t}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}$$

Hermite to Bézier

$$\mathbf{p}_0 = \mathbf{q}_0$$
$$\mathbf{p}_1 = \mathbf{q}_3$$
$$\mathbf{t}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$
$$\mathbf{t}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$



$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}$$

Bézier matrix

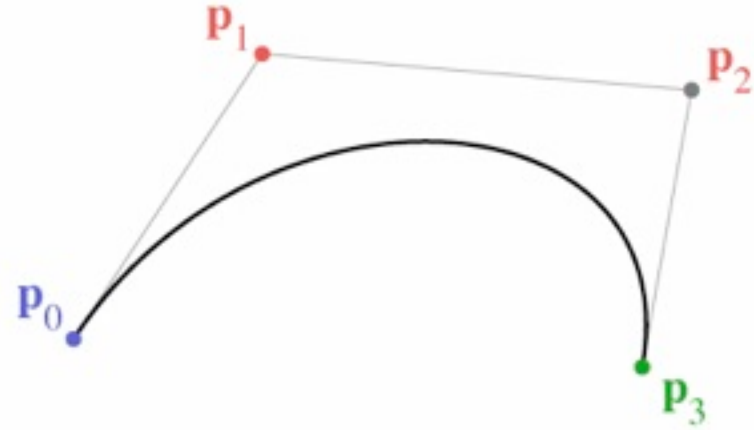
$$\mathbf{f}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

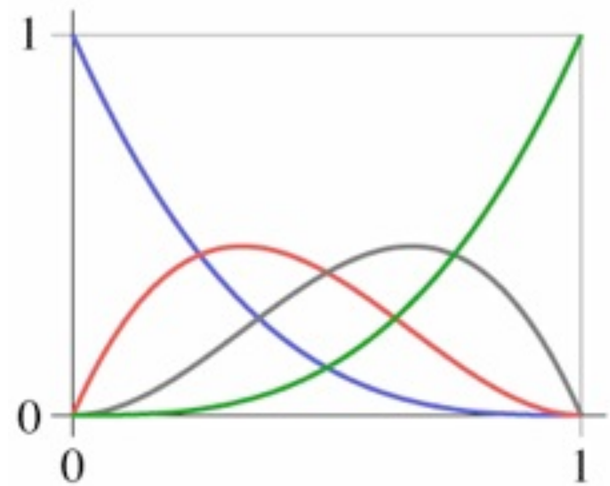
– note that these are the Bernstein polynomials

$$b_{n,k}(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

and that defines Bézier curves for any degree

Bézier basis



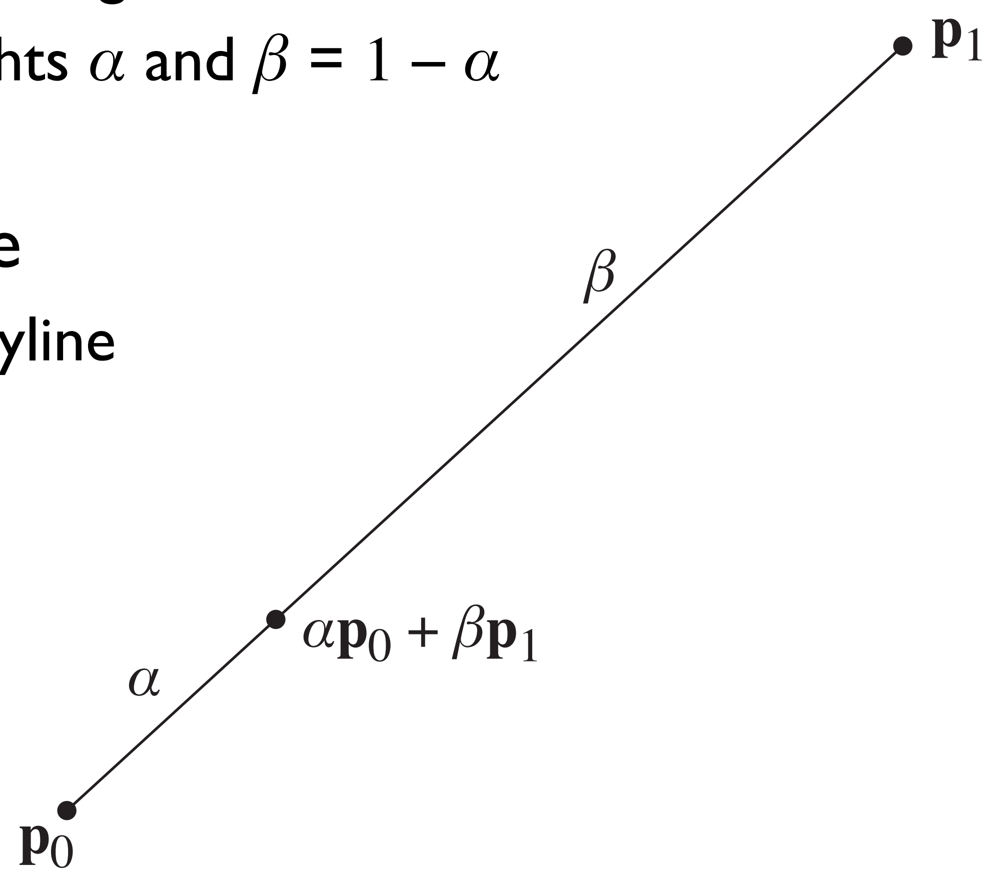


Another way to Bézier segments

- A really boring spline segment: f(t) = p0
  - it only has one control point
  - the curve stays at that point for the whole time
- Only good for building a *piecewise constant* spline
  - a.k.a. a set of points

Another way to Bézier segments

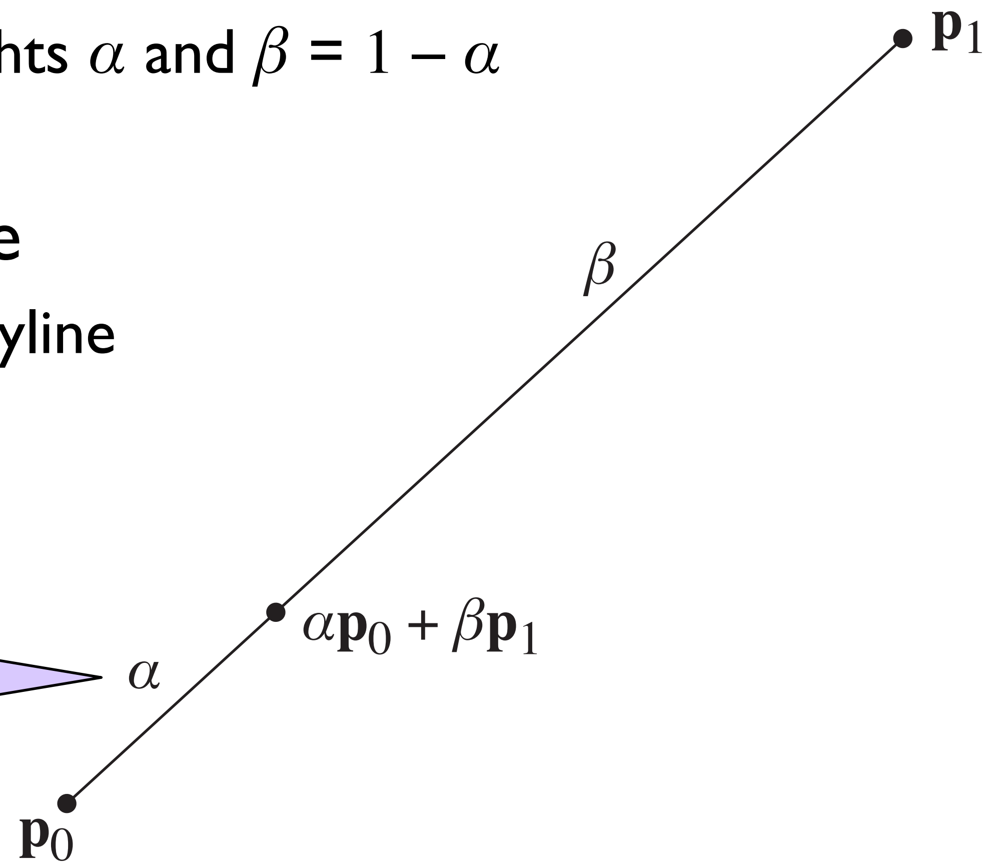
- A piecewise linear spline segment
  - two control points per segment
  - blend them with weights α and $\beta = 1 - \alpha$
- Good for building a piecewise linear spline
  - a.k.a. a polygon or polyline



Another way to Bézier segments

- A piecewise linear spline segment
  - two control points per segment
  - blend them with weights α and $\beta = 1 - \alpha$
- Good for building a piecewise linear spline
  - a.k.a. a polygon or polyline

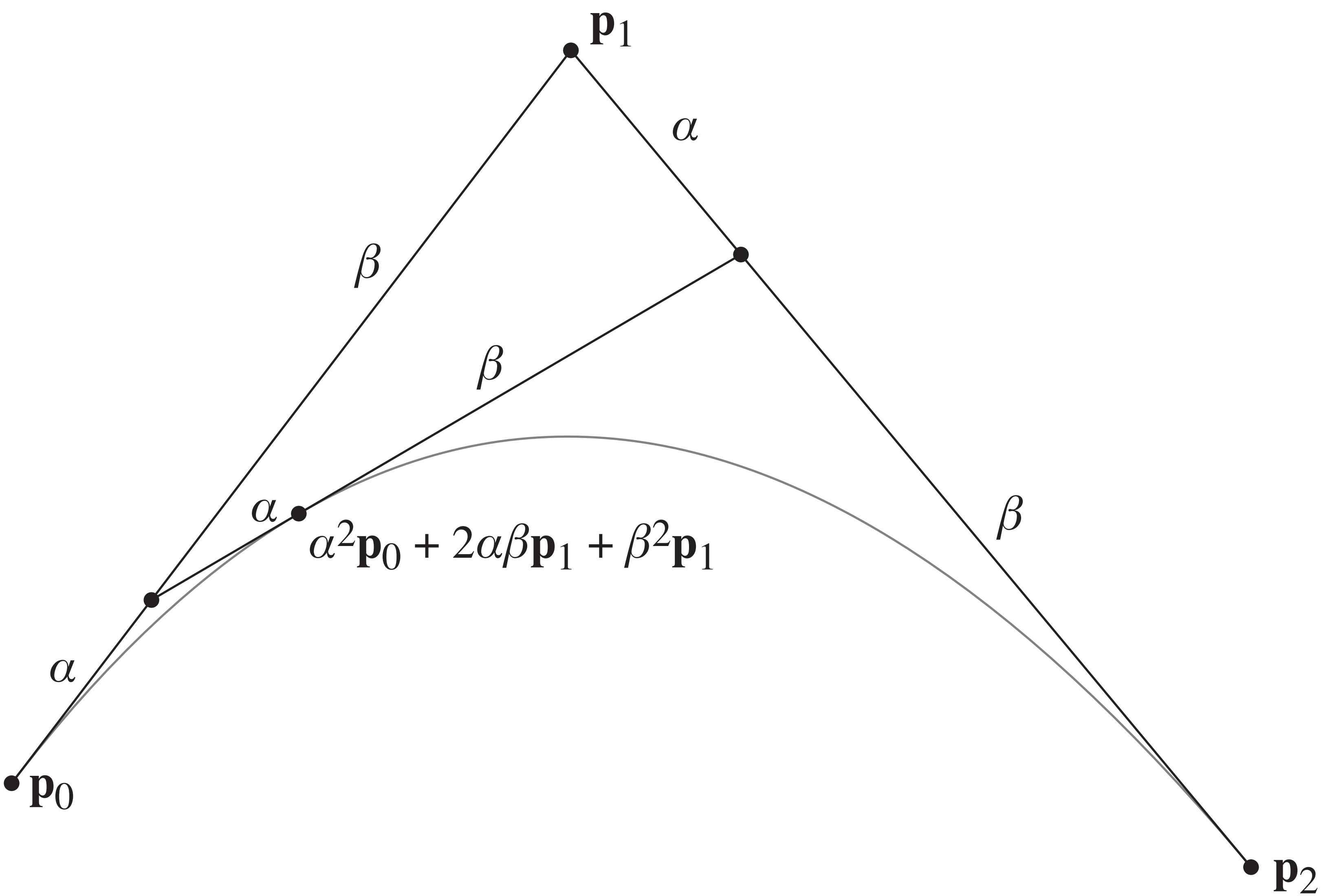
These labels show the **weights**, not the **distances**.



Another way to Bézier segments

- A linear blend of two piecewise linear segments
  - three control points now
  - interpolate on both segments using α and β
  - blend the results with the same weights
- makes a quadratic spline segment
  - finally, a curve!

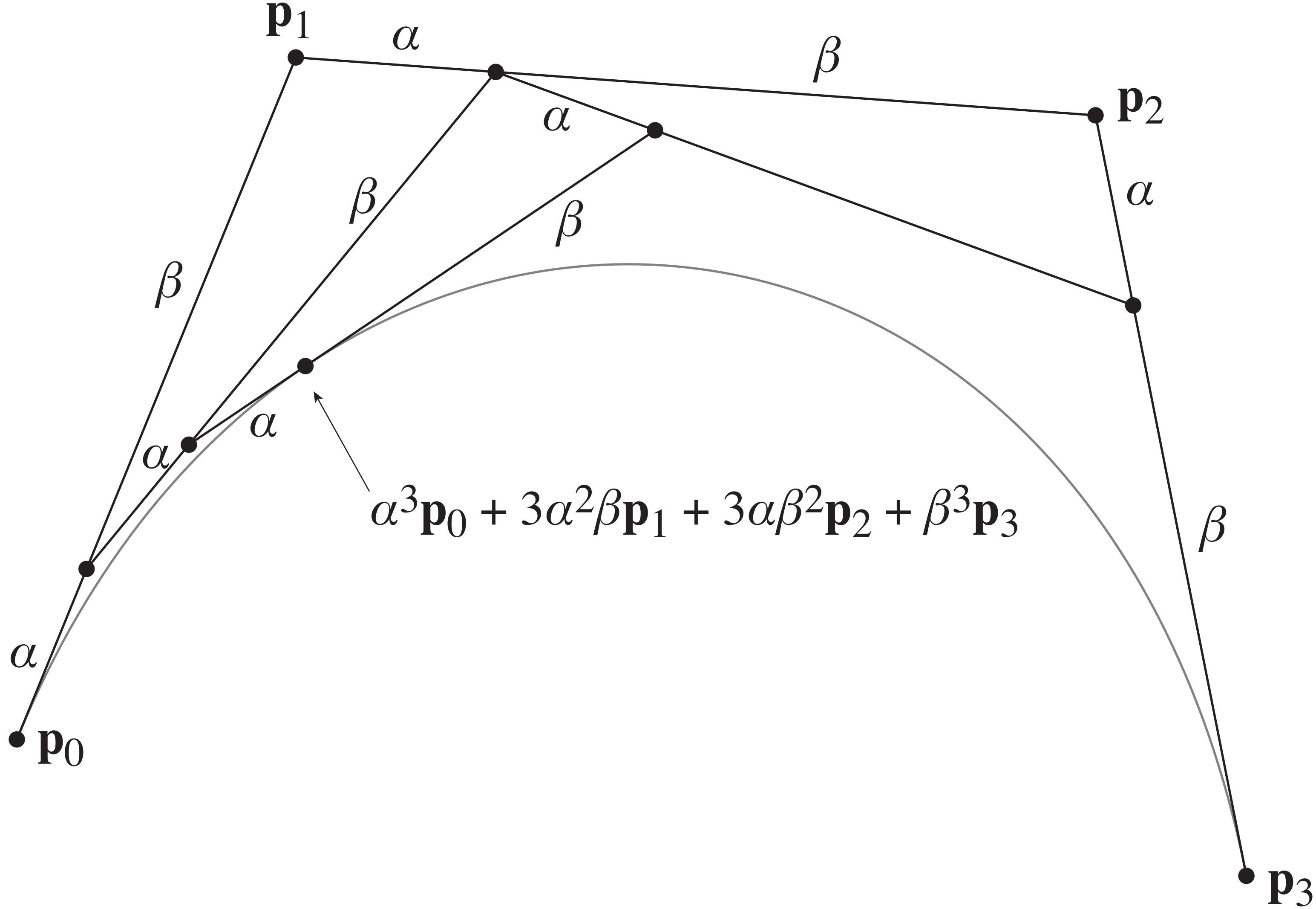
$$\begin{aligned}
\mathbf{p}_{1,0} &= \alpha\mathbf{p}_0 + \beta\mathbf{p}_1 \\
\mathbf{p}_{1,1} &= \alpha\mathbf{p}_1 + \beta\mathbf{p}_2 \\
\mathbf{p}_{2,0} &= \alpha\mathbf{p}_{1,0} + \beta\mathbf{p}_{1,1} \\
&= \alpha\alpha\mathbf{p}_0 + \alpha\beta\mathbf{p}_1 + \beta\alpha\mathbf{p}_1 + \beta\beta\mathbf{p}_2 \\
&= \alpha^2\mathbf{p}_0 + 2\alpha\beta\mathbf{p}_1 + \beta^2\mathbf{p}_2
\end{aligned}$$


$\mathbf{p}_1$
α
β
β
α
$\alpha^2\mathbf{p}_0 + 2\alpha\beta\mathbf{p}_1 + \beta^2\mathbf{p}_1$
β
α
$\mathbf{p}_0$
$\mathbf{p}_2$

Another way to Bézier segments

- Cubic segment: blend of two quadratic segments
  - four control points now (overlapping sets of 3)
  - interpolate on each quadratic using α and β
  - blend the results with the same weights
- makes a cubic spline segment
  - this is the familiar one for graphics—but you can keep going

$$\begin{aligned}
\mathbf{p}_{3,0} &= \alpha \mathbf{p}_{2,0} + \beta \mathbf{p}_{2,1} \\
&= \alpha\alpha\alpha \mathbf{p}_0 + \alpha\alpha\beta \mathbf{p}_1 + \alpha\beta\alpha \mathbf{p}_1 + \alpha\beta\beta \mathbf{p}_2 \\
&\quad \beta\alpha\alpha \mathbf{p}_1 + \beta\alpha\beta \mathbf{p}_2 + \beta\beta\alpha \mathbf{p}_2 + \beta\beta\beta \mathbf{p}_3 \\
&= \alpha^3 \mathbf{p}_0 + 3\alpha^2\beta \mathbf{p}_1 + 3\alpha\beta^2 \mathbf{p}_2 + \beta^3 \mathbf{p}_3
\end{aligned}$$

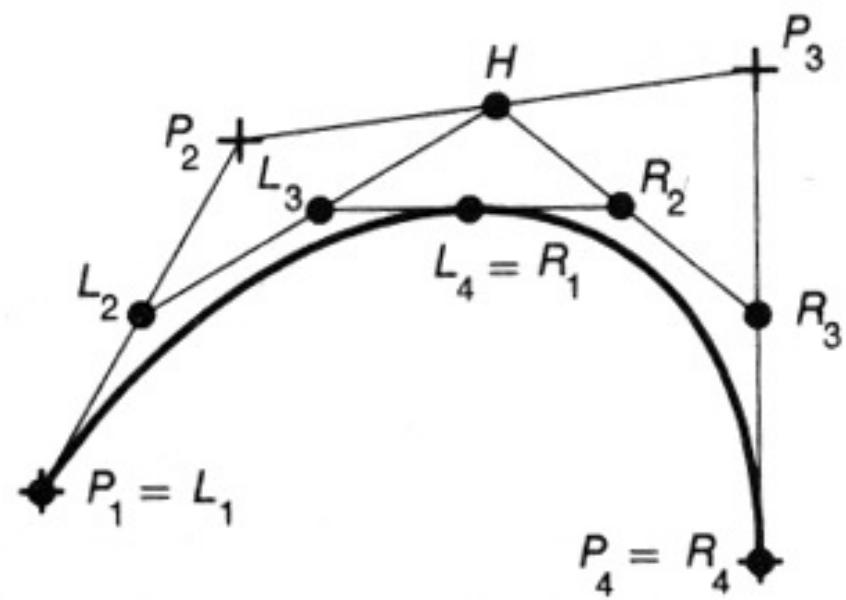

$\mathbf{p}_1$
α
β
$\mathbf{p}_2$
α
β
α
β
β
β
α
α
α
$\alpha^3\mathbf{p}_0 + 3\alpha^2\beta\mathbf{p}_1 + 3\alpha\beta^2\mathbf{p}_2 + \beta^3\mathbf{p}_3$
β
α
$\mathbf{p}_0$
$\mathbf{p}_3$

de Casteljau's algorithm

- A recurrence for computing points on Bézier spline segments:

$$\mathbf{p}_{0,i} = \mathbf{p}_i$$
$$\mathbf{p}_{n,i} = \alpha \mathbf{p}_{n-1,i} + \beta \mathbf{p}_{n-1,i+1}$$

- Cool additional feature: also subdivides the segment into two shorter ones



[FvDFH]

Cubic Bézier splines

- Very widely used type, especially in 2D
  - e.g. it is a primitive in PostScript/PDF
- Can represent smooth curves with corners
- Nice de Casteljau recurrence for evaluation
- Can easily add points at any position
- Illustrator demo

Spline Curves

Chaining spline segments

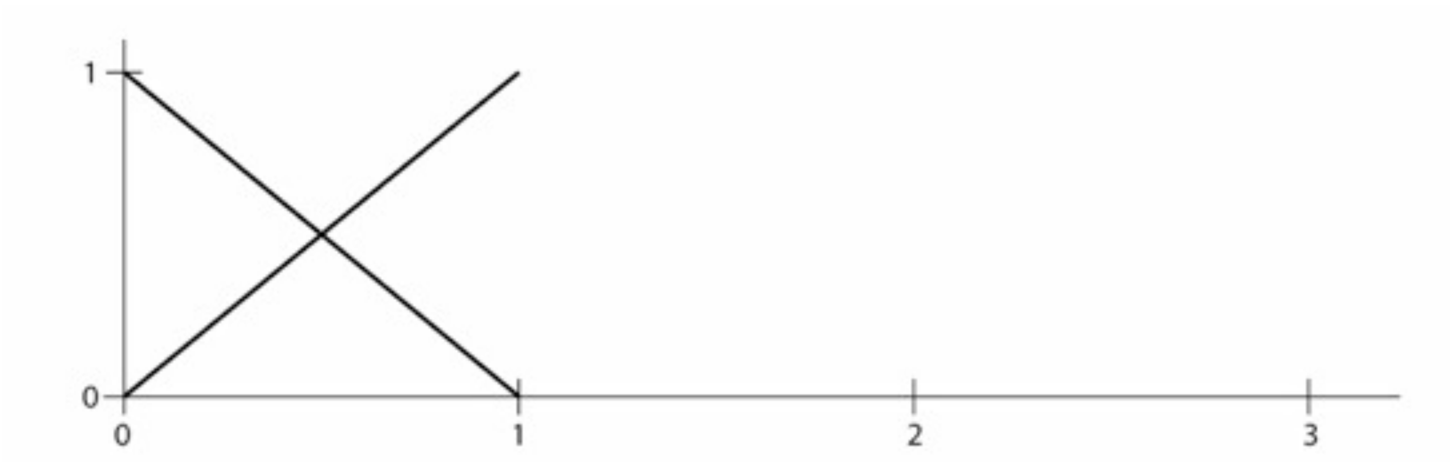
- Can only do so much with a single polynomial
- Can use these functions as segments of a longer curve
  - curve from t = 0 to t = 1 defined by first segment
  - curve from t = 1 to t = 2 defined by second segment

$$\mathbf{f}(t) = \mathbf{f}_i(t - i) \quad \text{for } i \leq t \leq i + 1$$

- To avoid discontinuity, match derivatives at junctions
  - this produces a C^1 curve

Trivial example: piecewise linear

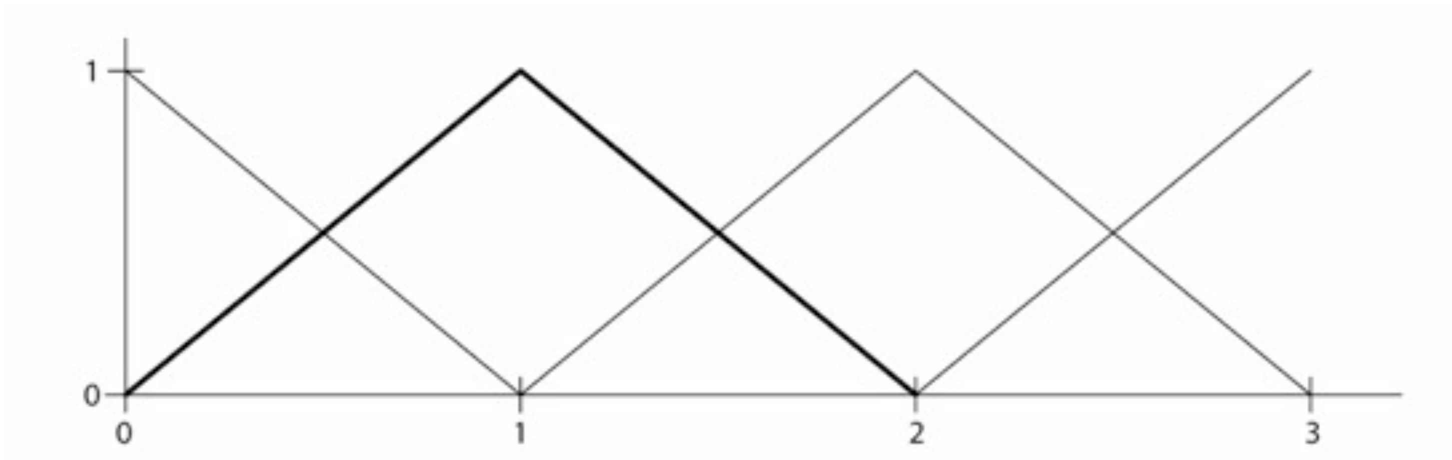
- Basis function formulation: "function times point"
  - basis functions: contribution of each point as *t* changes



  - can think of them as blending functions glued together
  - this is just like a reconstruction filter!

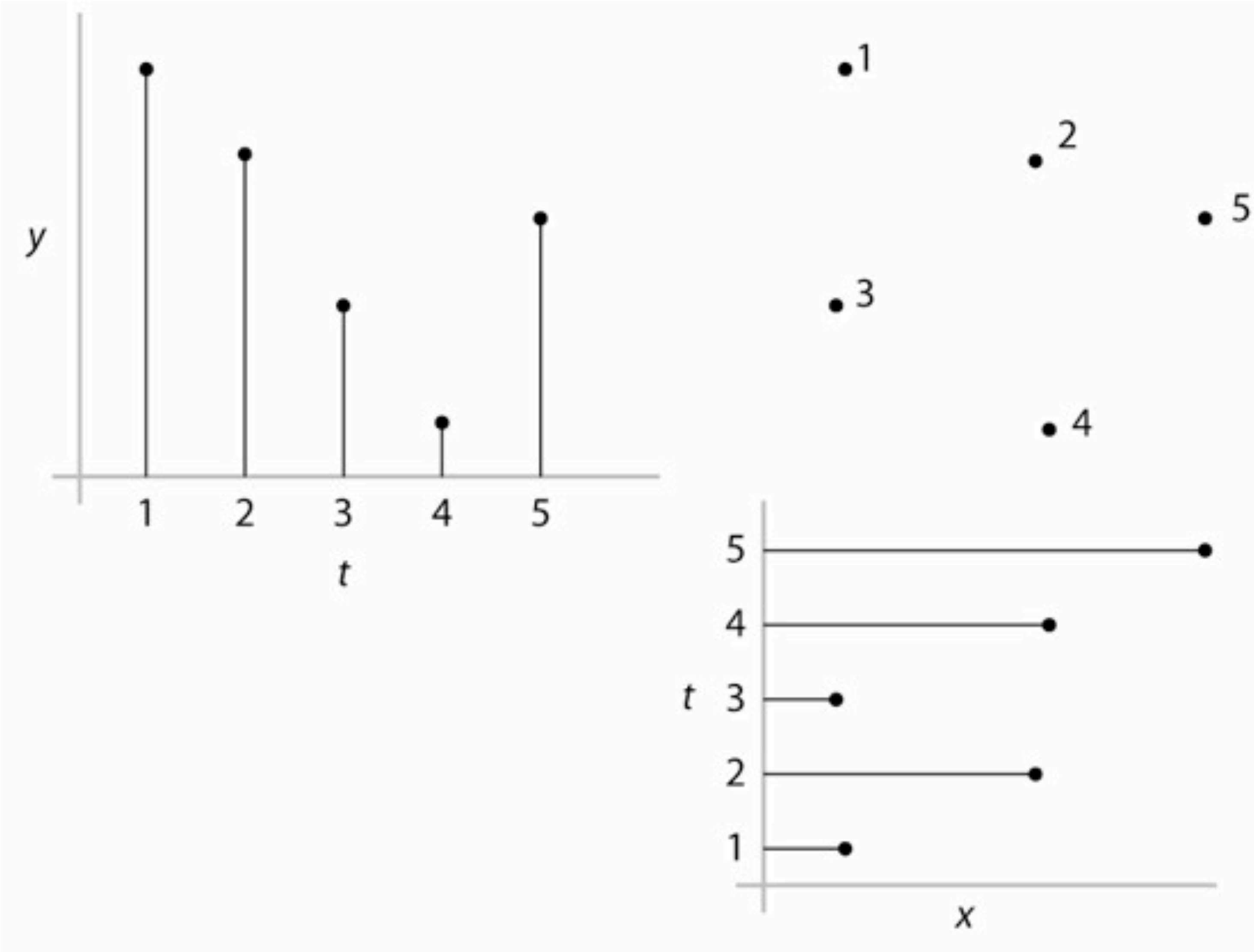
Trivial example: piecewise linear

- Basis function formulation: "function times point"
  - basis functions: contribution of each point as *t* changes

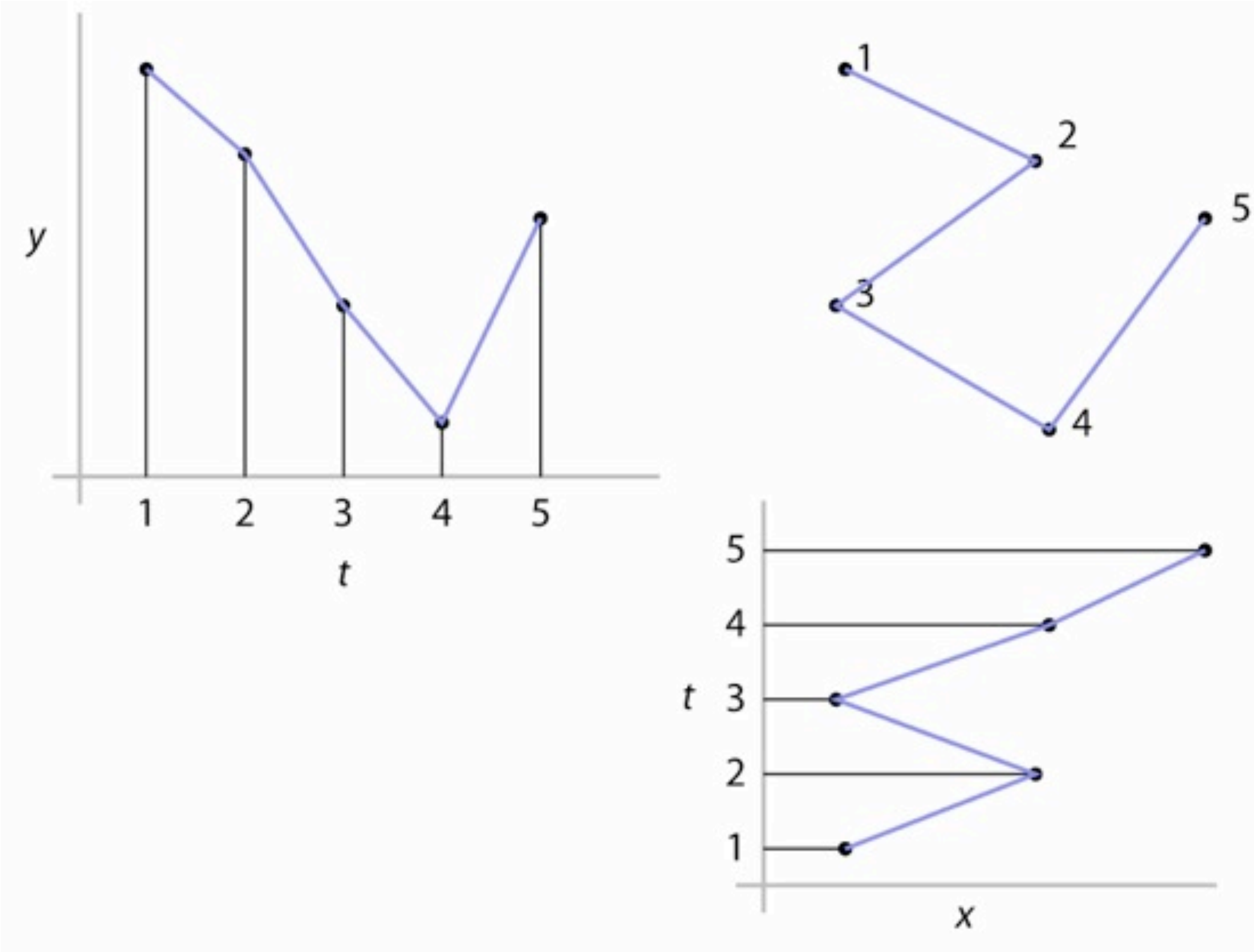


  - can think of them as blending functions glued together
  - this is just like a reconstruction filter!

Splines as reconstruction



Splines as reconstruction

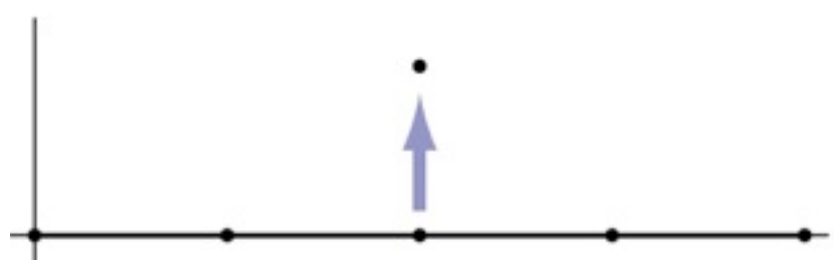


Seeing the basis functions

- Basis functions of a spline are revealed by how the curve changes in response to a change in one control
  - to get a graph of the basis function, start with the curve laid out in a straight, constant-speed line
    - what are $x(t)$ and $y(t)$?
  - then move one control straight up

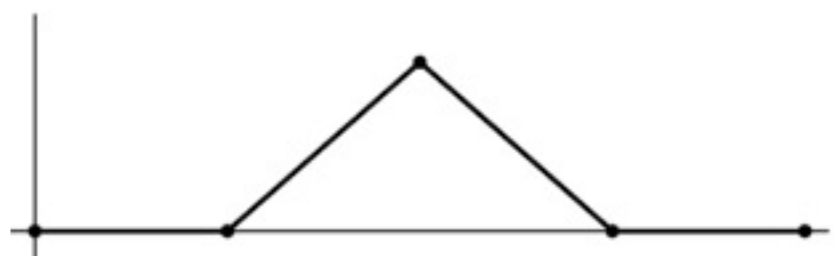
Seeing the basis functions

- Basis functions of a spline are revealed by how the curve changes in response to a change in one control
  - to get a graph of the basis function, start with the curve laid out in a straight, constant-speed line
    - what are $x(t)$ and $y(t)$?
  - then move one control straight up



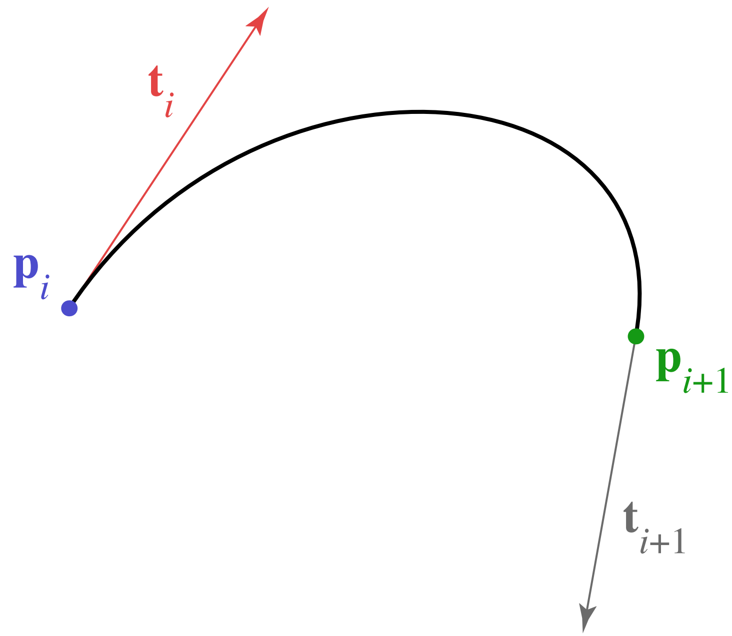
Seeing the basis functions

- Basis functions of a spline are revealed by how the curve changes in response to a change in one control
  - to get a graph of the basis function, start with the curve laid out in a straight, constant-speed line
    - what are $x(t)$ and $y(t)$?
  - then move one control straight up



Hermite splines

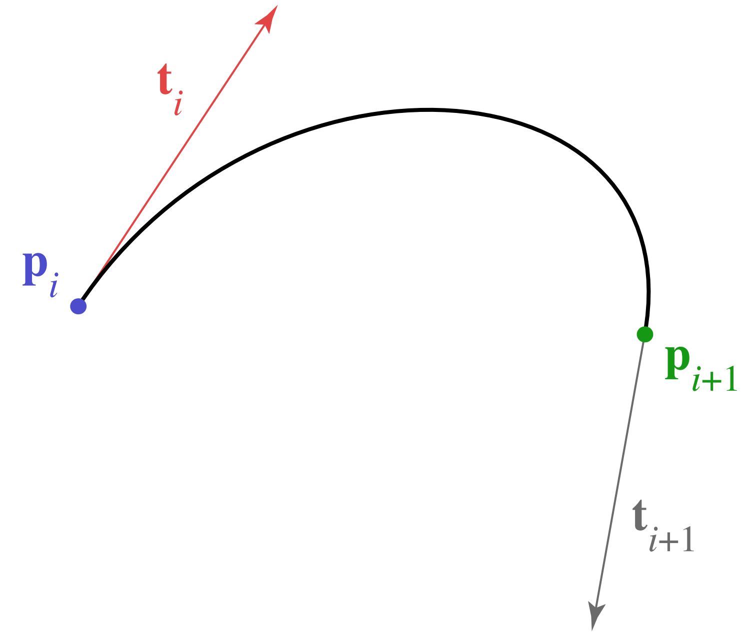
- Constraints are endpoints and endpoint tangents

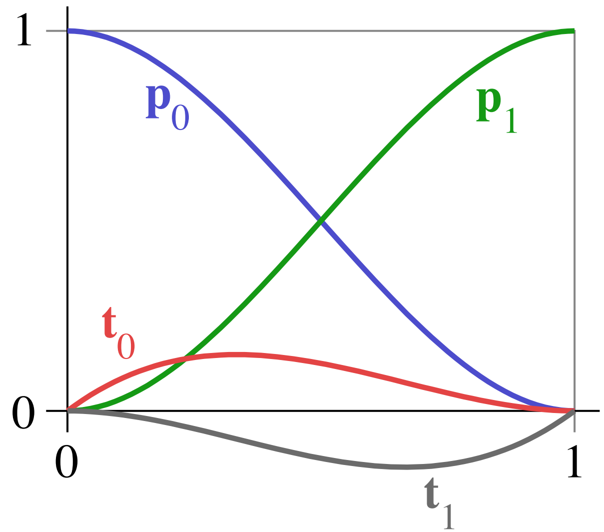


$$\mathbf{f}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 2 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}'_0 \\ \mathbf{p}'_1 \end{bmatrix}$$

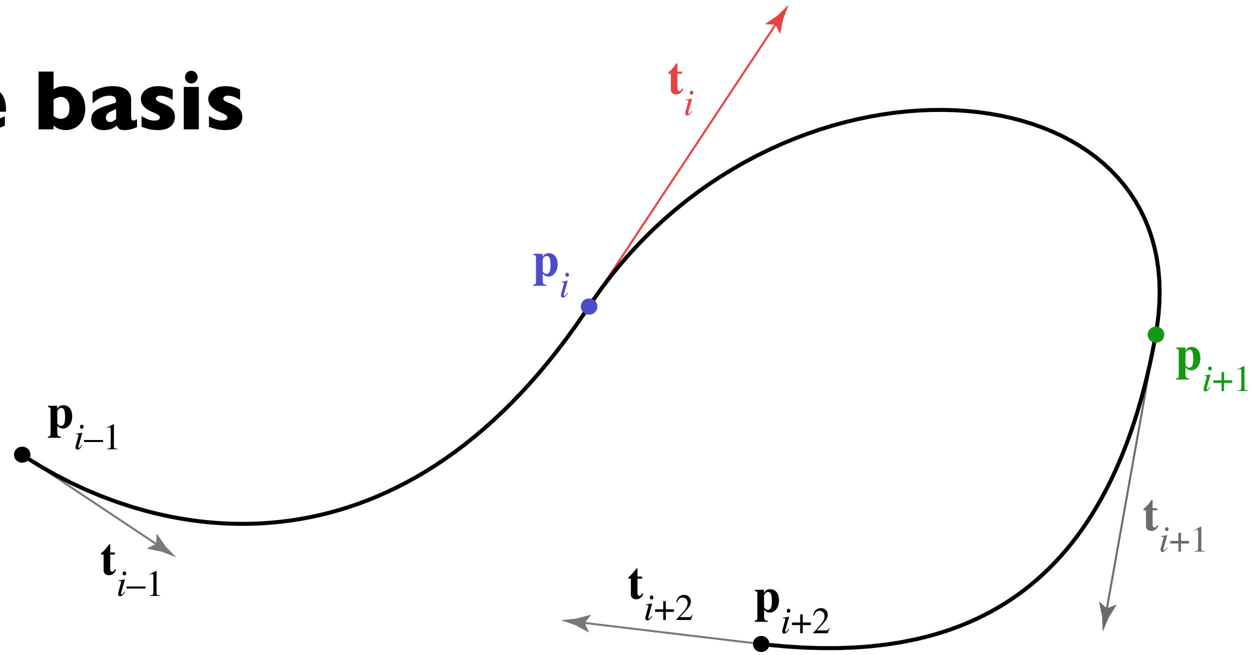
Hermite basis

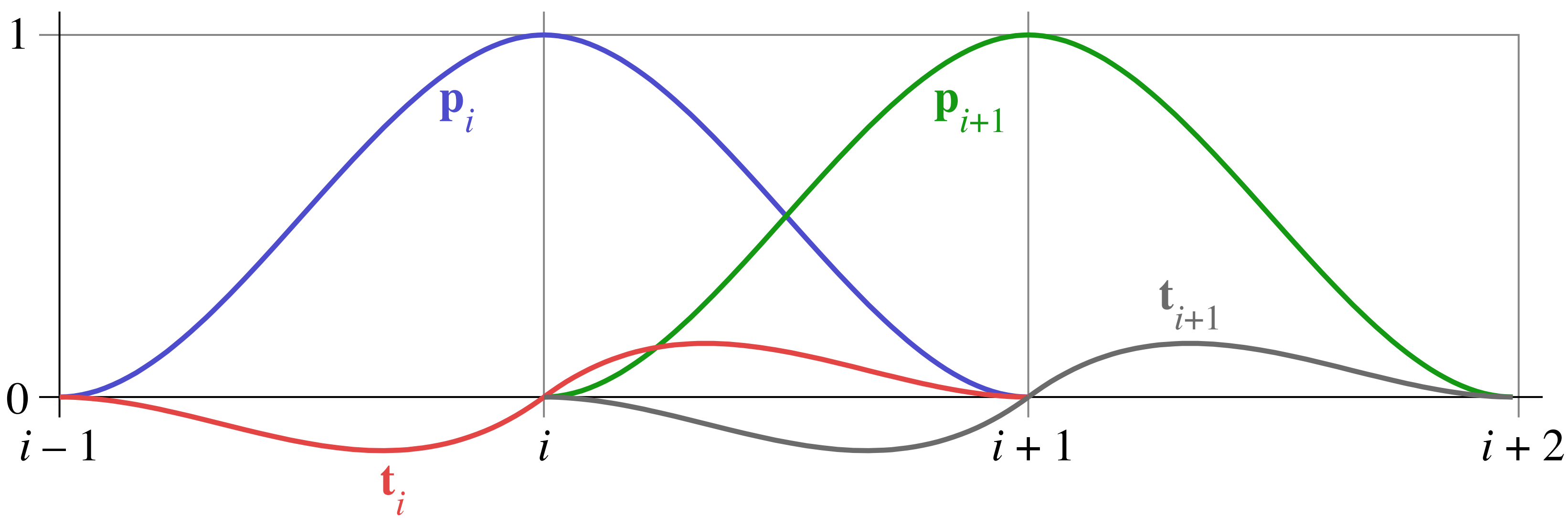
Hermite basis



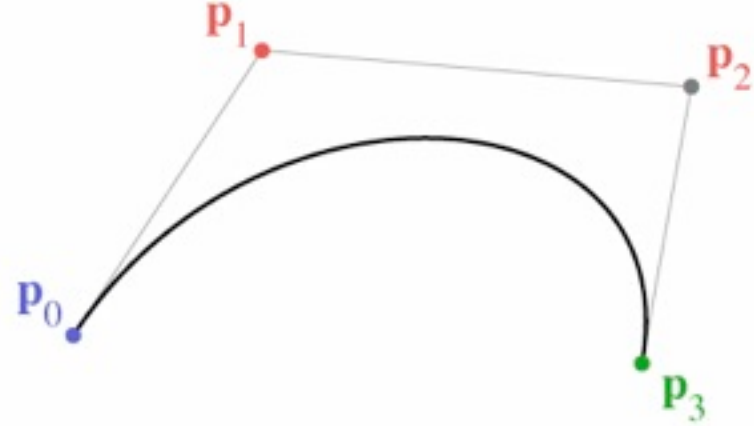


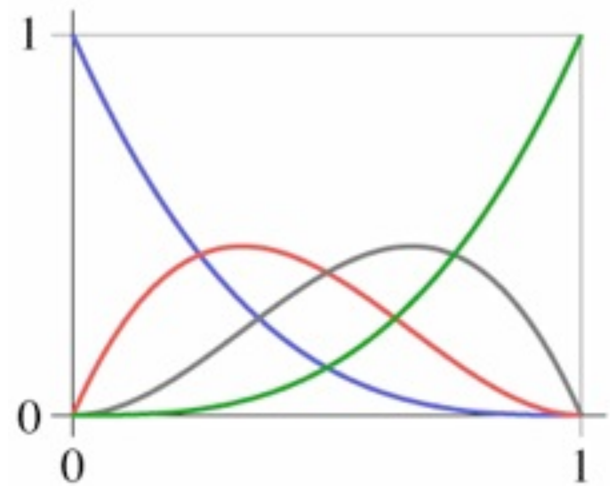
Hermite basis





Bézier basis



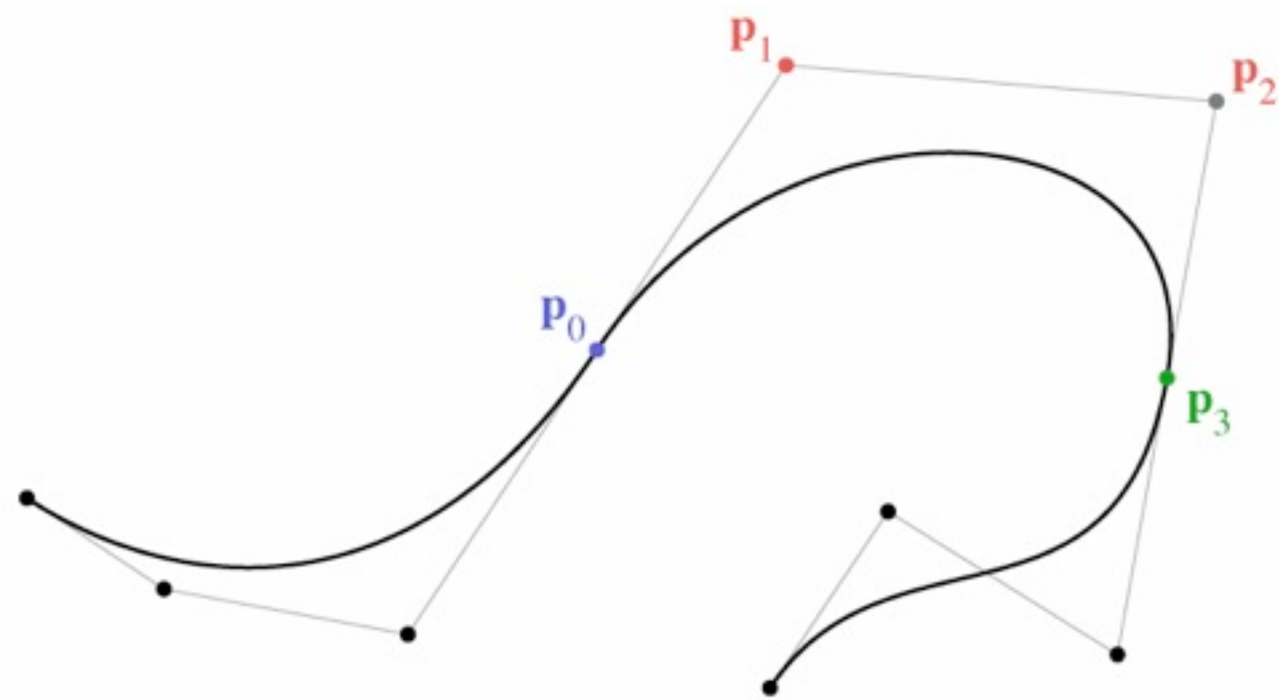


Chaining Bézier splines

- No continuity built in
- Achieve C^1 using collinear control points

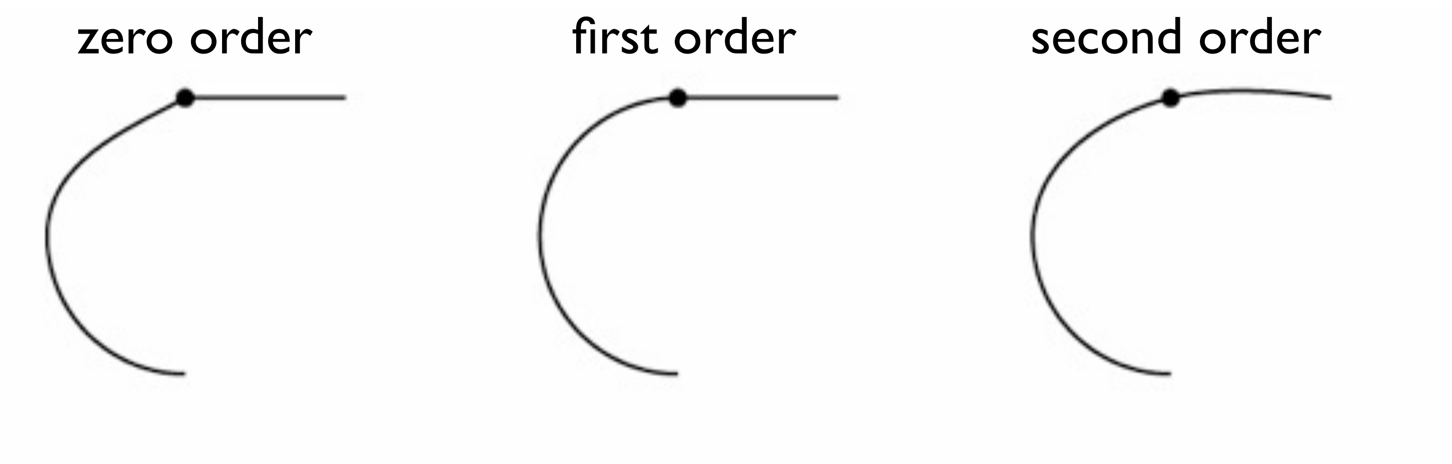
Chaining Bézier splines

- No continuity built in
- Achieve C^1 using collinear control points



Continuity

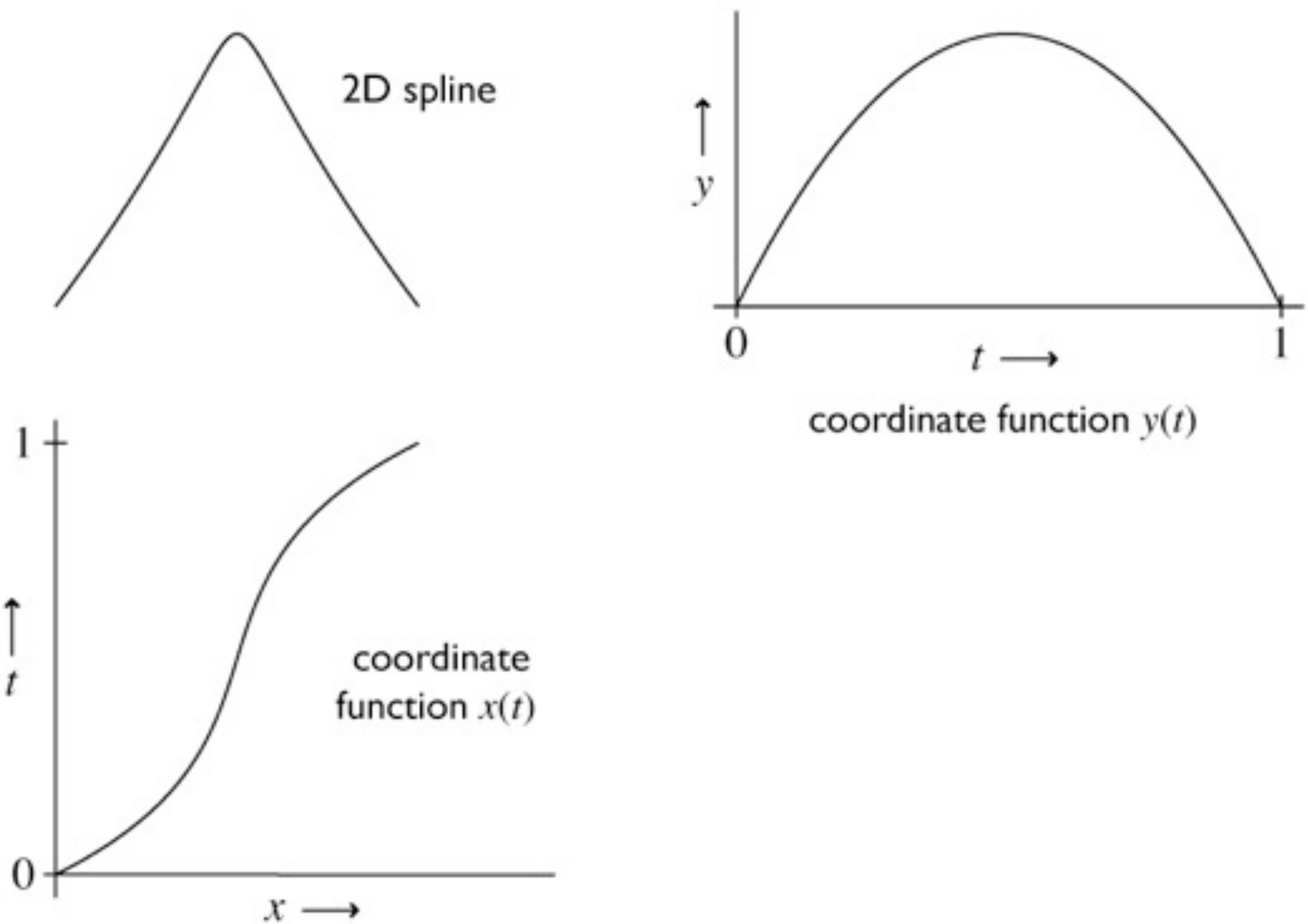
- Smoothness can be described by degree of continuity
  - zero-order (C^0): position matches from both sides
  - first-order (C^1): tangent matches from both sides
  - second-order (C^2): curvature matches from both sides
  - G^n vs. C^n



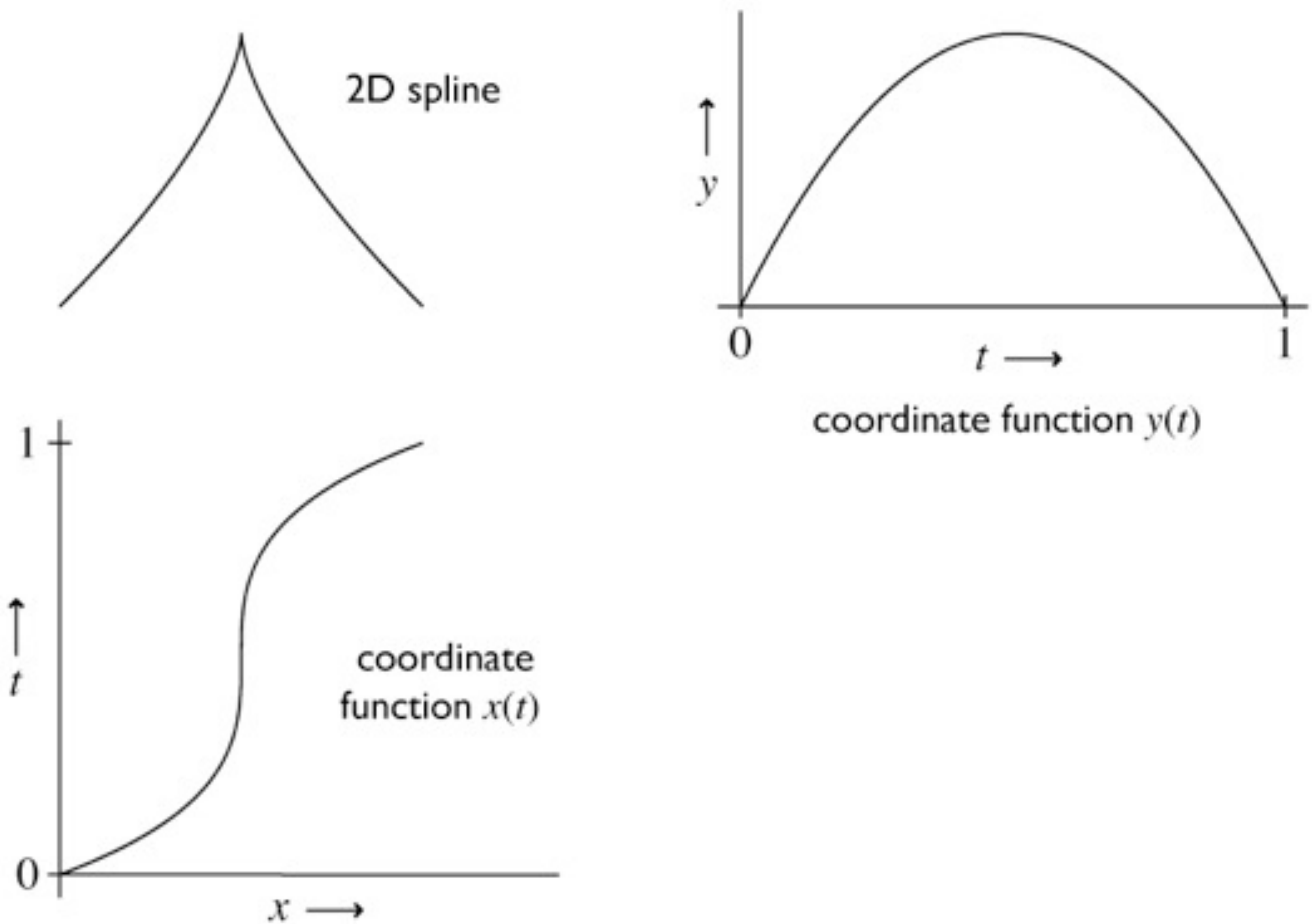
Continuity

- Parametric continuity (*C*) of spline is continuity of coordinate functions
- Geometric continuity (*G*) is continuity of the curve itself
- Neither form of continuity is guaranteed by the other
  - Can be C^1 but not G^1 when $\mathbf{p}(t)$ comes to a halt (next slide)
  - Can be G^1 but not C^1 when the tangent vector changes length abruptly

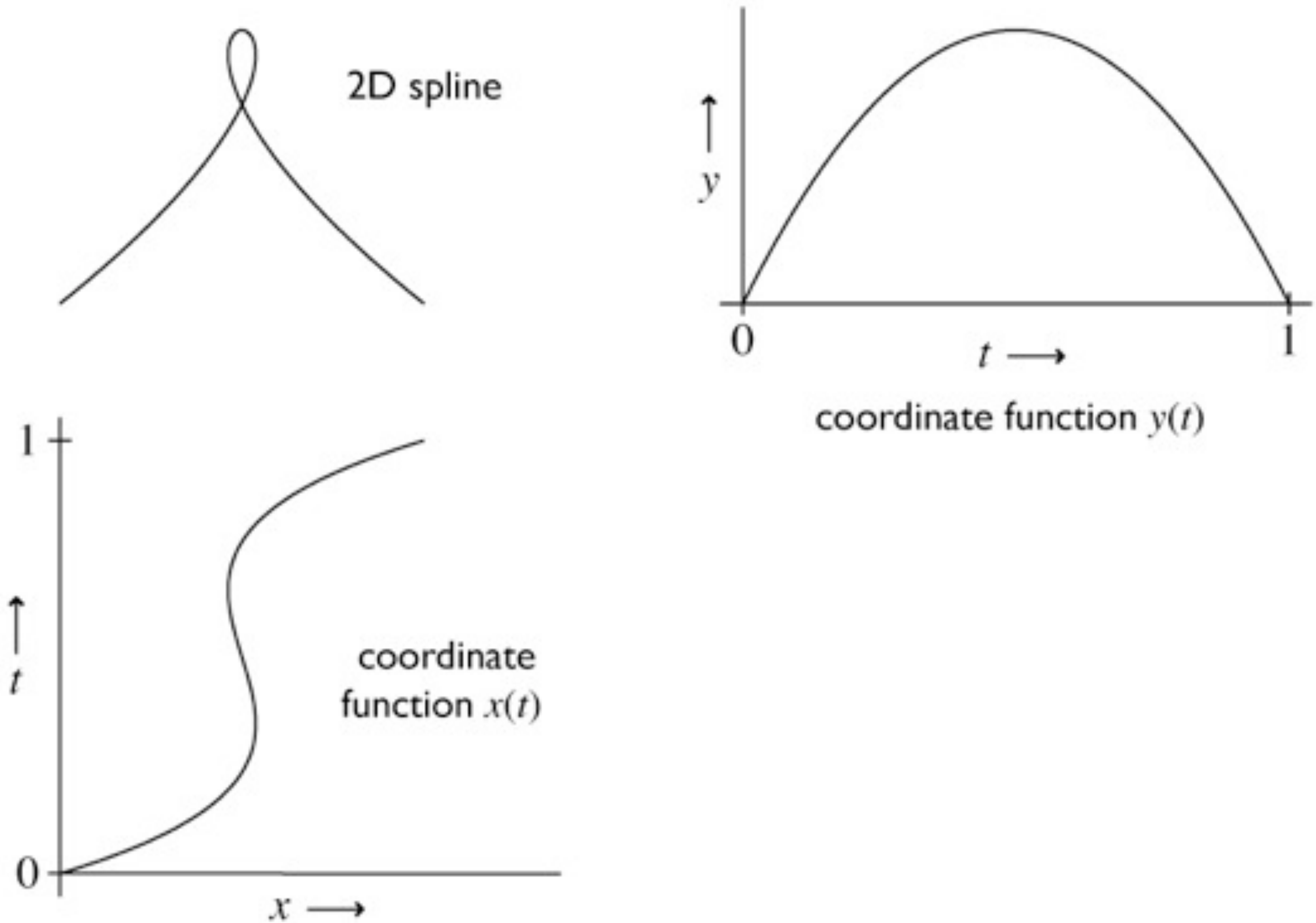
Geometric vs. parametric continuity



Geometric vs. parametric continuity



Geometric vs. parametric continuity

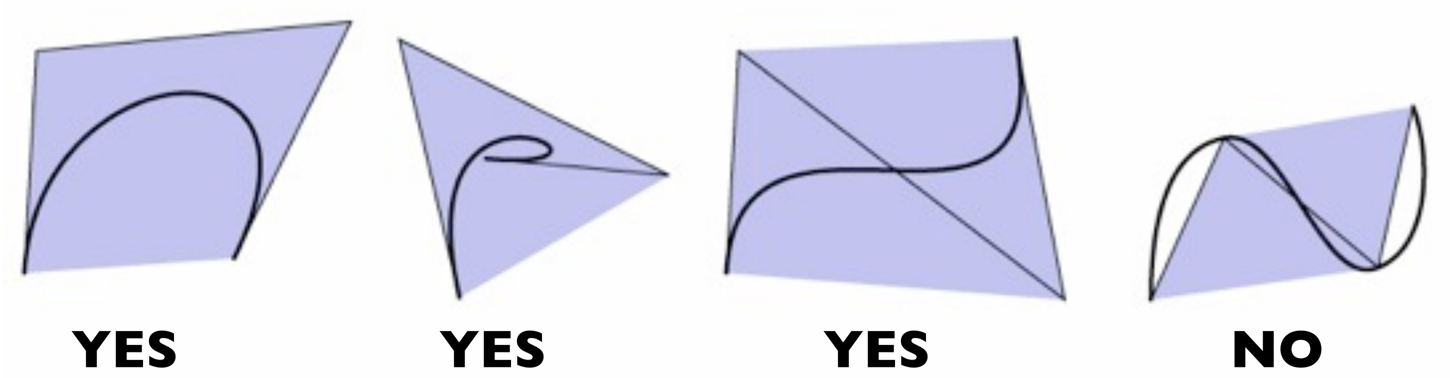


Control

- Local control
  - changing control point only affects a limited part of spline
  - without this, splines are very difficult to use
  - many likely formulations lack this
    - natural spline
    - polynomial fits

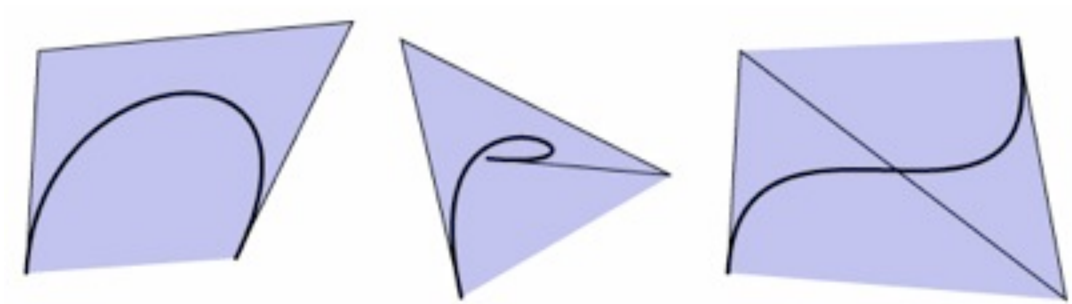
Control

- Convex hull property
  - convex hull = smallest convex region containing points
    - think of a rubber band around some pins
  - some splines stay inside convex hull of control points
    - make clipping, culling, picking, etc. simpler



Convex hull

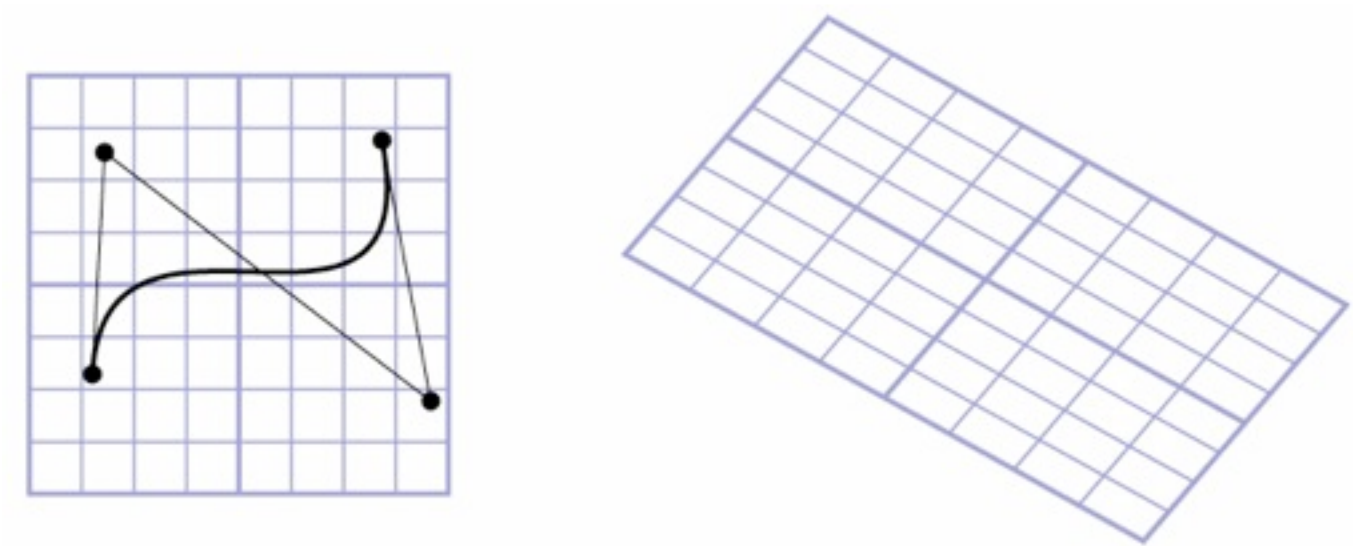
- If basis functions are all positive, the spline has the convex hull property
  – we're still requiring them to sum to 1



  – if any basis function is ever negative, no convex hull prop.
    - proof: take the other three points at the same place

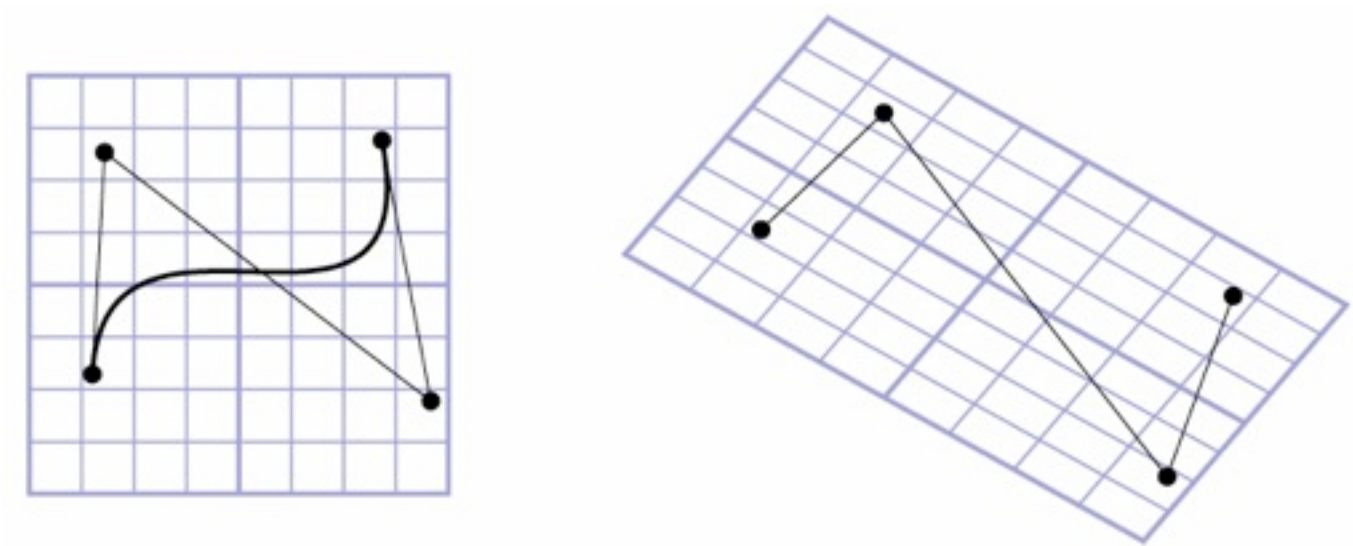
Affine invariance

- Transforming the control points is the same as transforming the curve
  - true for all commonly used splines
  - extremely convenient in practice…



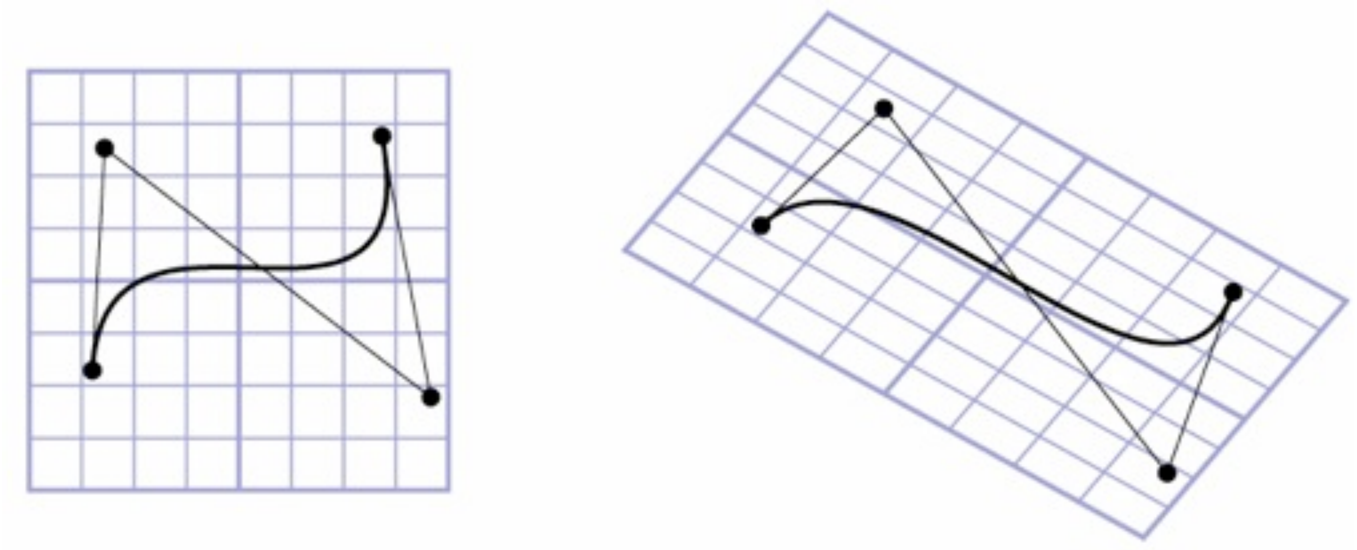
Affine invariance

- Transforming the control points is the same as transforming the curve
  - true for all commonly used splines
  - extremely convenient in practice…



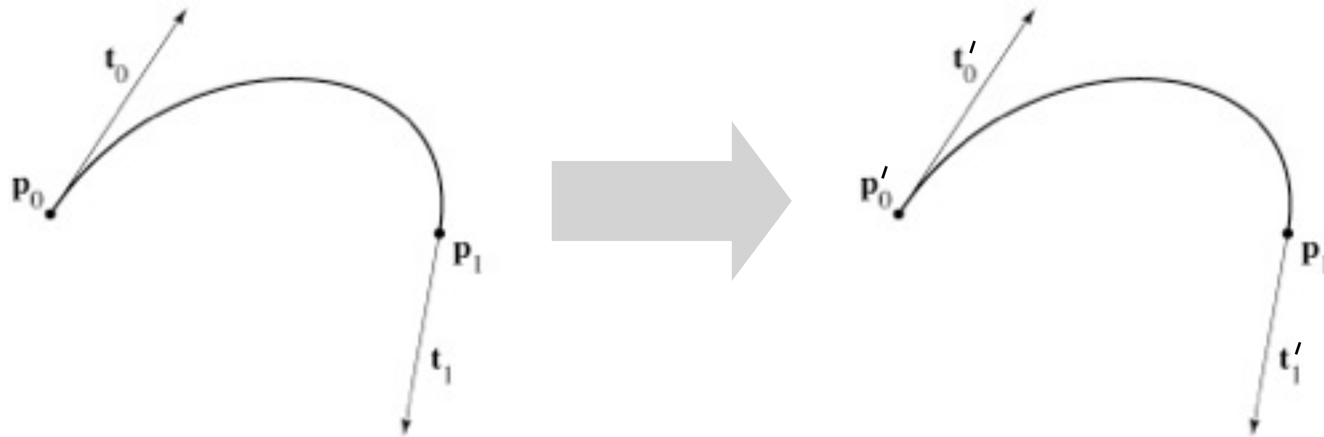
Affine invariance

- Transforming the control points is the same as transforming the curve
  - true for all commonly used splines
  - extremely convenient in practice…



Affine invariance

- Basis functions associated with points should always sum to 1



$$\begin{aligned}
\mathbf{p}(t) &= b_0\mathbf{p}_0 + b_1\mathbf{p}_1 + b_2\mathbf{v}_0 + b_3\mathbf{v}_1 \\
\mathbf{p}'(t) &= b_0(\mathbf{p}_0 + \mathbf{u}) + b_1(\mathbf{p}_1 + \mathbf{u}) + b_2\mathbf{v}_0 + b_3\mathbf{v}_1 \\
&= b_0\mathbf{p}_0 + b_1\mathbf{p}_1 + b_2\mathbf{v}_0 + b_3\mathbf{v}_1 + (b_0 + b_1)\mathbf{u} \\
&= \mathbf{p}(t) + \mathbf{u}
\end{aligned}$$

Chaining spline segments

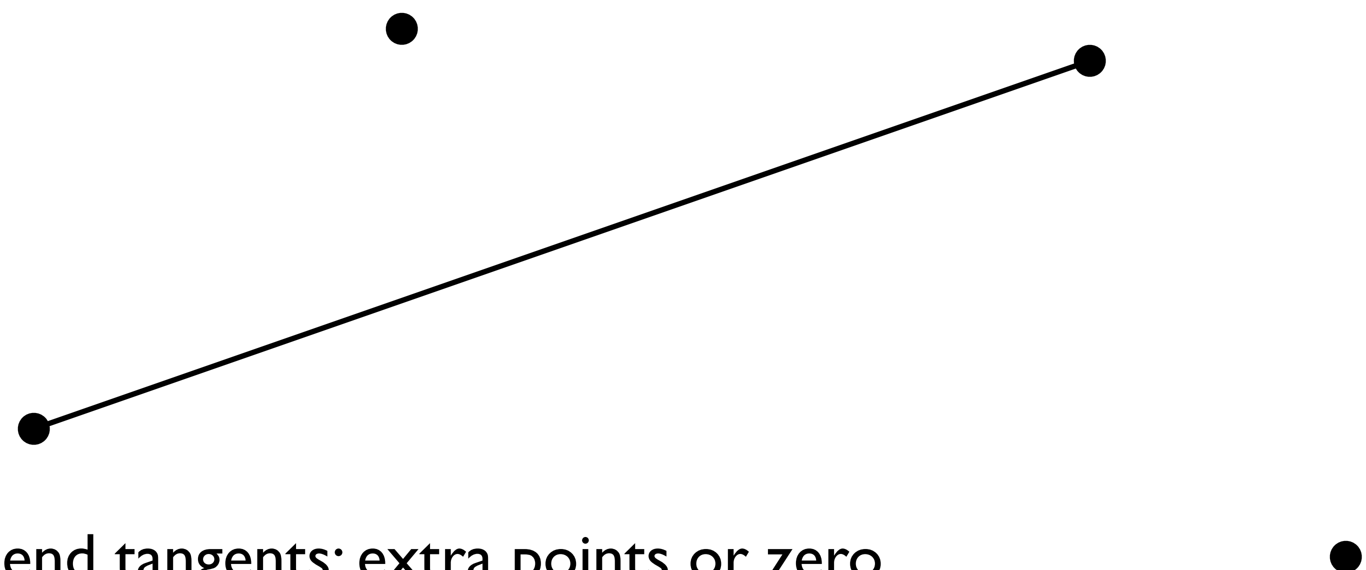
- Hermite curves are convenient because they can be made long easily
- Bézier curves are convenient because their controls are all points
  - but it is fussy to maintain continuity constraints
  - and they interpolate every 3rd point, which is a little odd
- We derived Bézier from Hermite by defining tangents from control points
  - a similar construction leads to the interpolating *Catmull-Rom* spline

Hermite to Catmull-Rom

- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points
  - end tangents: extra points or zero

Hermite to Catmull-Rom

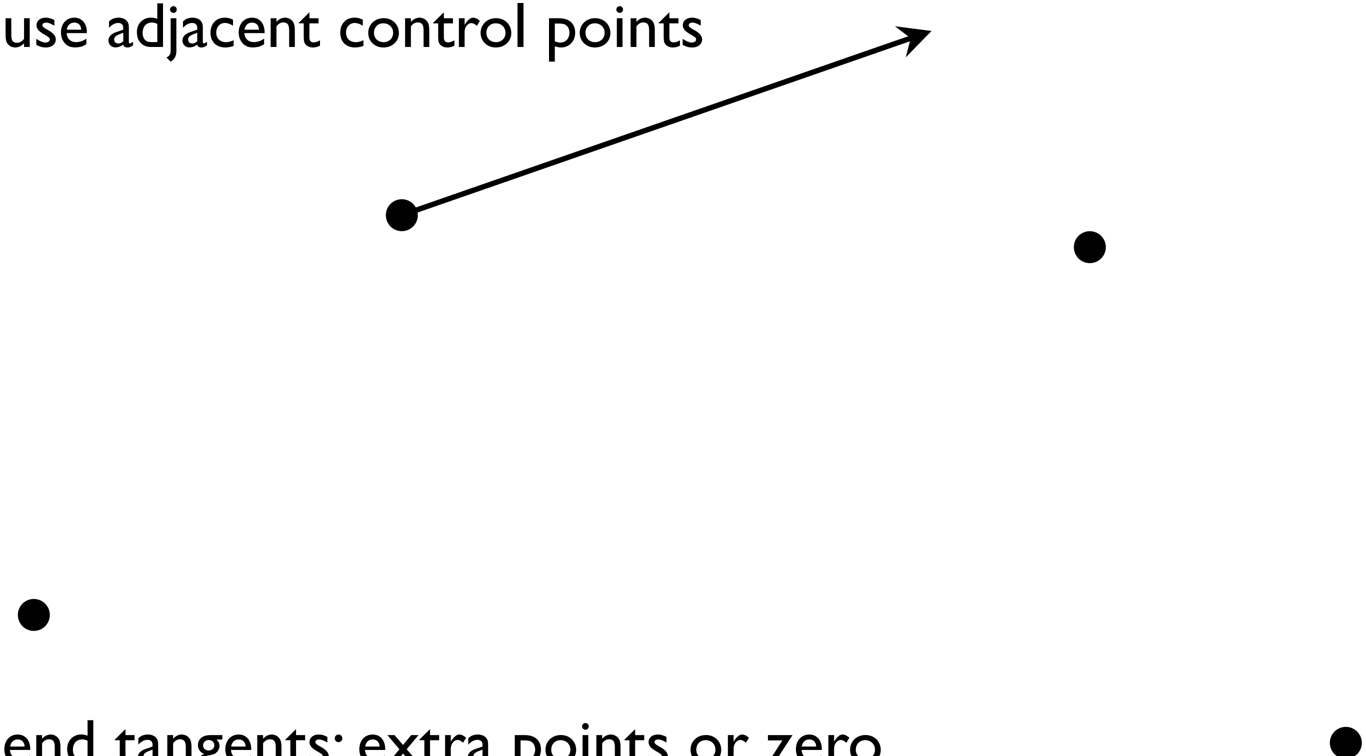
- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points



  - end tangents: extra points or zero

Hermite to Catmull-Rom

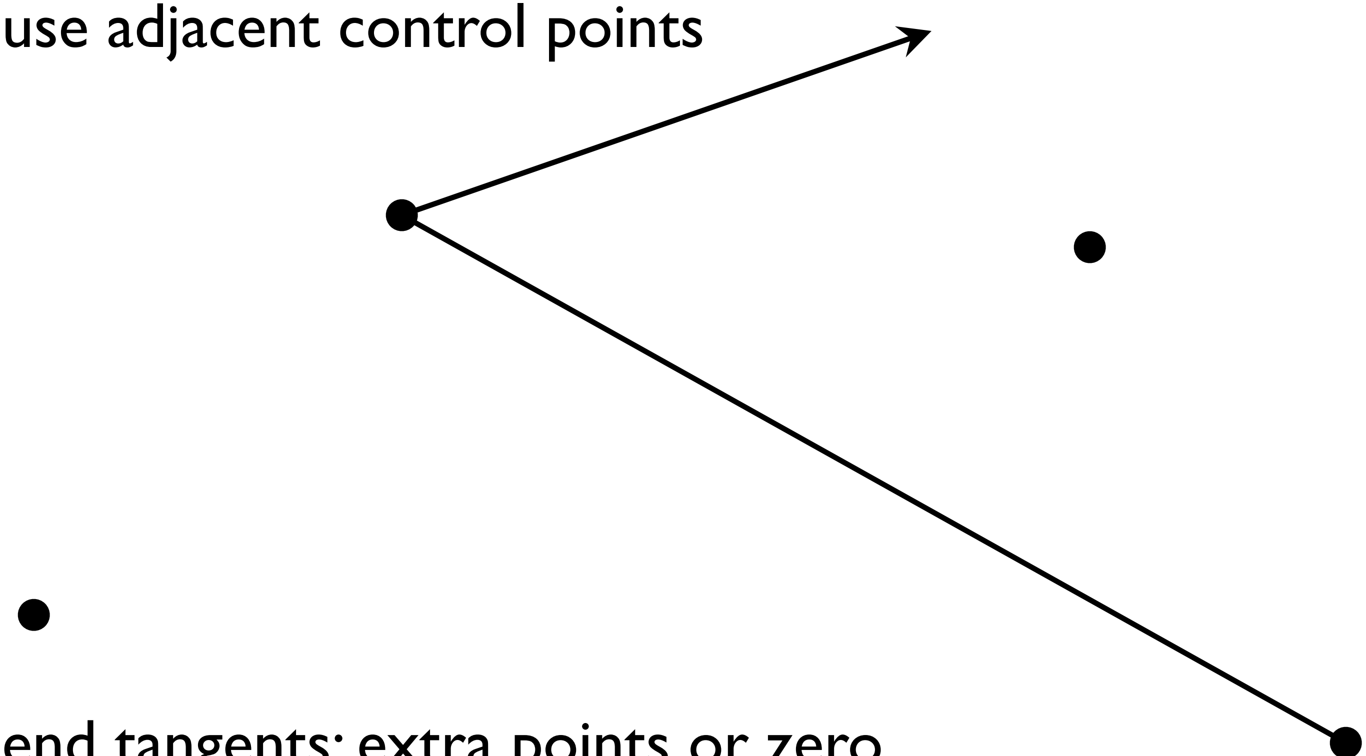
- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points



  - end tangents: extra points or zero

Hermite to Catmull-Rom

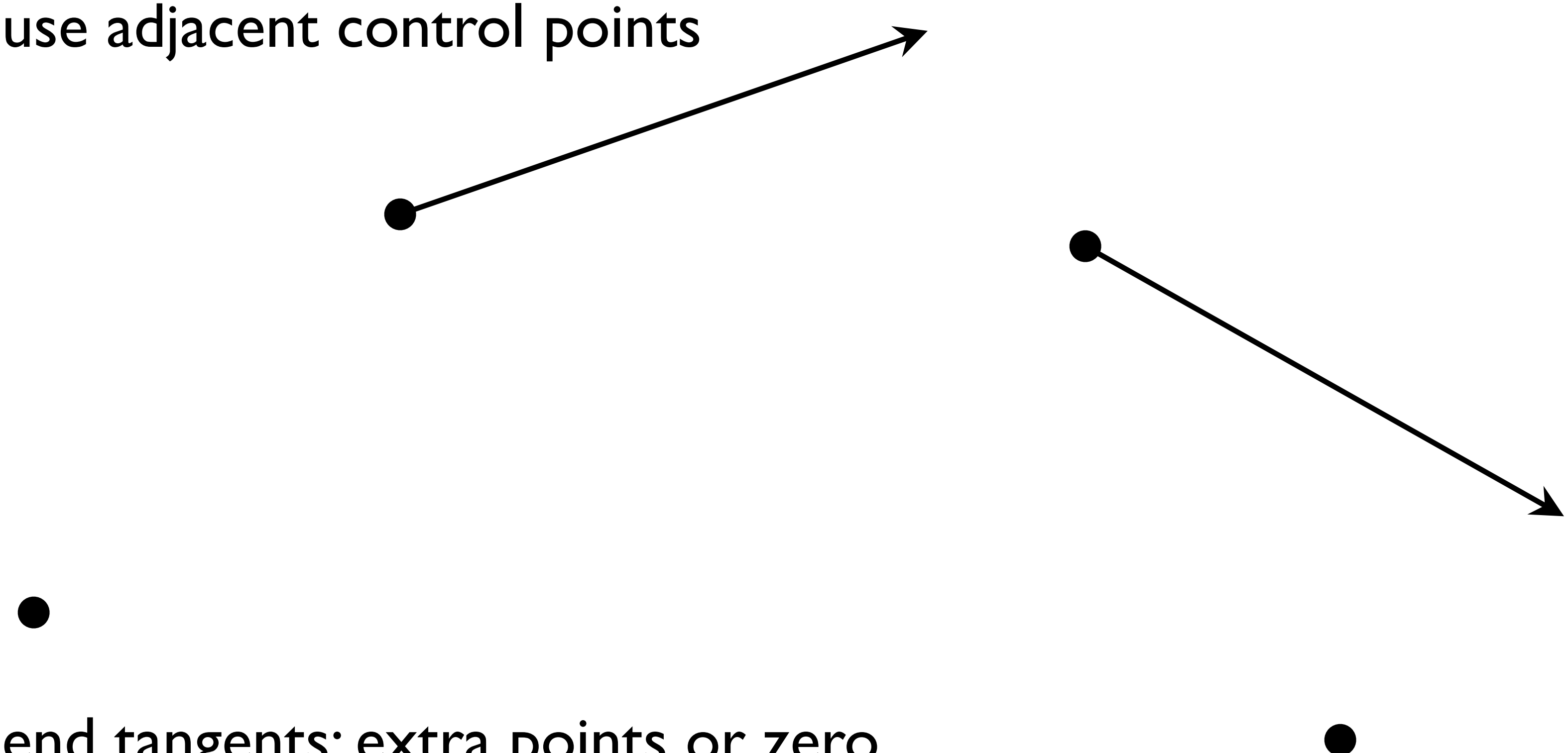
- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points



  - end tangents: extra points or zero

Hermite to Catmull-Rom

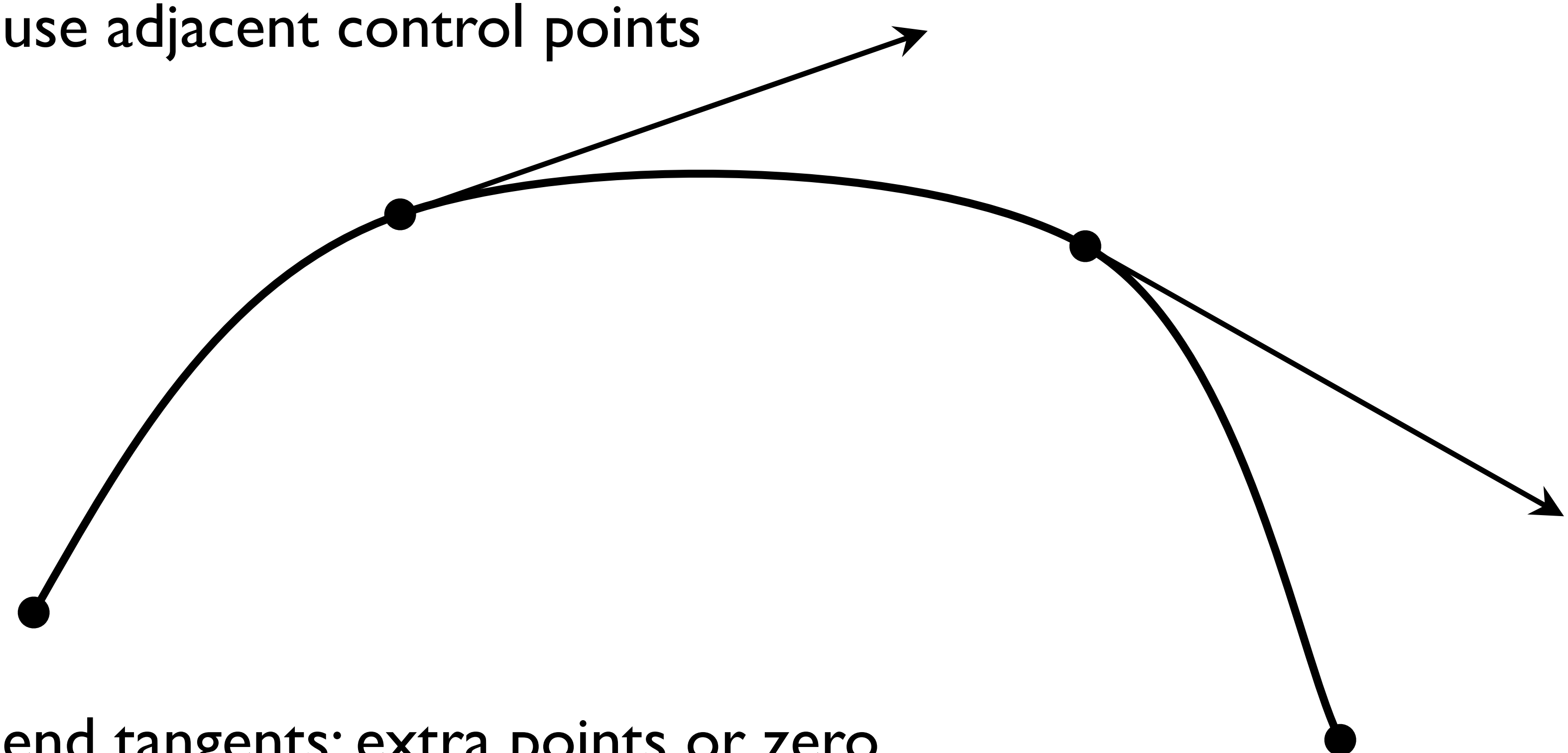
- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points



  - end tangents: extra points or zero

Hermite to Catmull-Rom

- Have not yet seen any interpolating splines
- Would like to define tangents automatically
  - use adjacent control points



  - end tangents: extra points or zero

Hermite to Catmull-Rom

- Tangents are $(\mathbf{p}_{k+1} - \mathbf{p}_{k-1}) / 2$
  - scaling based on same argument about collinear case

$$\mathbf{p}_0 = \mathbf{q}_k$$

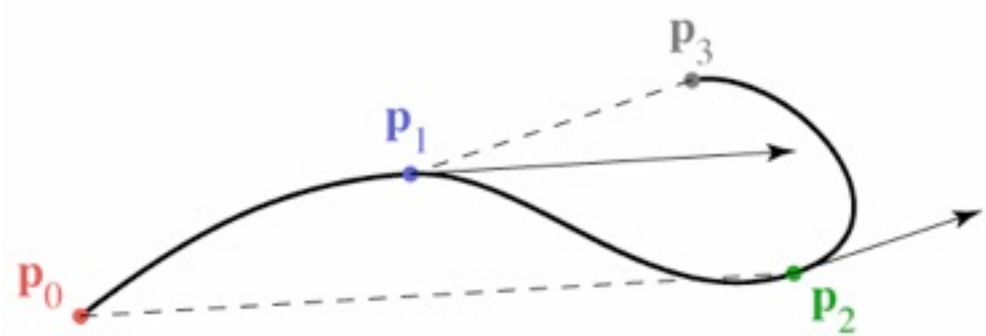
$$\mathbf{p}_1 = \mathbf{q}_k + 1$$

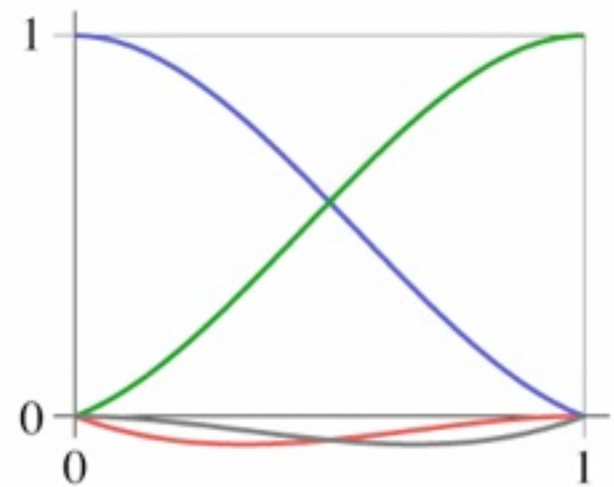
$$\mathbf{v}_0 = 0.5(\mathbf{q}_{k+1} - \mathbf{q}_{k-1})$$

$$\mathbf{v}_1 = 0.5(\mathbf{q}_{k+2} - \mathbf{q}_K)$$

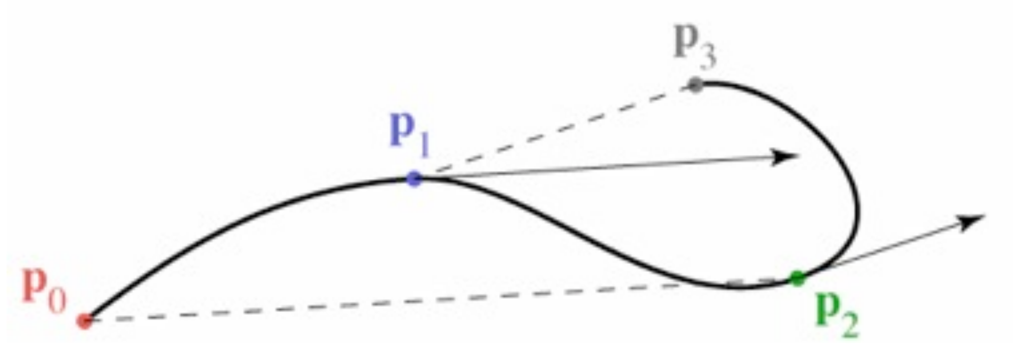
$$\begin{bmatrix}\mathbf{a}\\\mathbf{b}\\\mathbf{c}\\\mathbf{d}\end{bmatrix} = \begin{bmatrix}2 & -2 & 1 & 1\\-3 & 3 & -2 & -1\\0 & 0 & 1 & 0\\1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\-.5 & 0 & .5 & 0\\0 & -.5 & 0 & .5\end{bmatrix}\begin{bmatrix}\mathbf{q}_{k-1}\\\mathbf{q}_k\\\mathbf{q}_{k+1}\\\mathbf{q}_{k+2}\end{bmatrix}$$

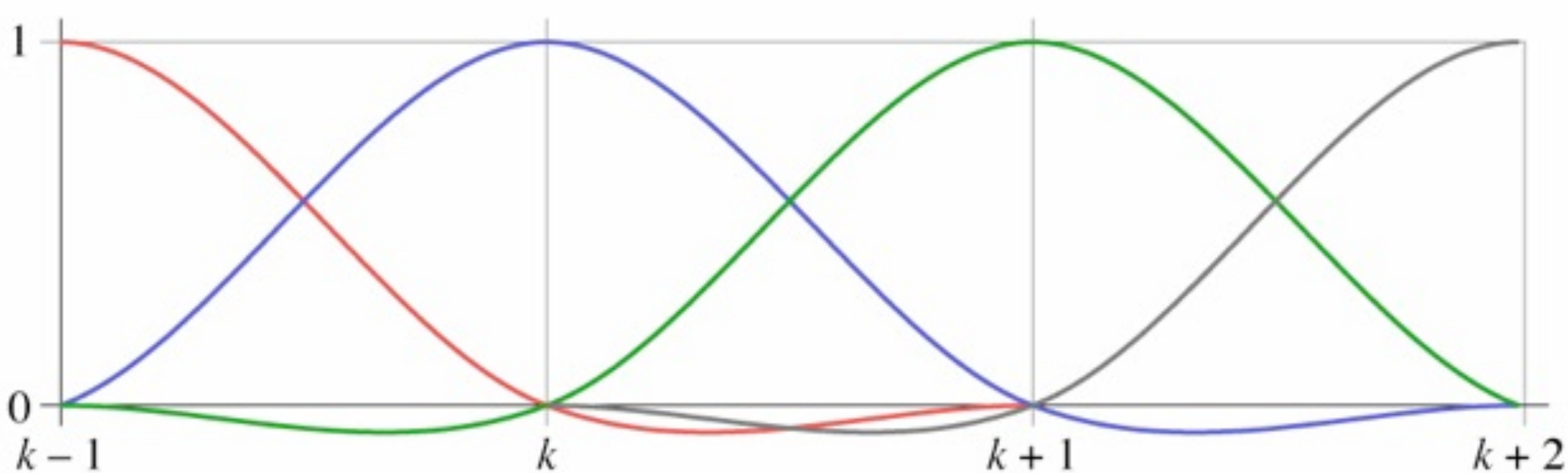
Catmull-Rom basis





Catmull-Rom basis





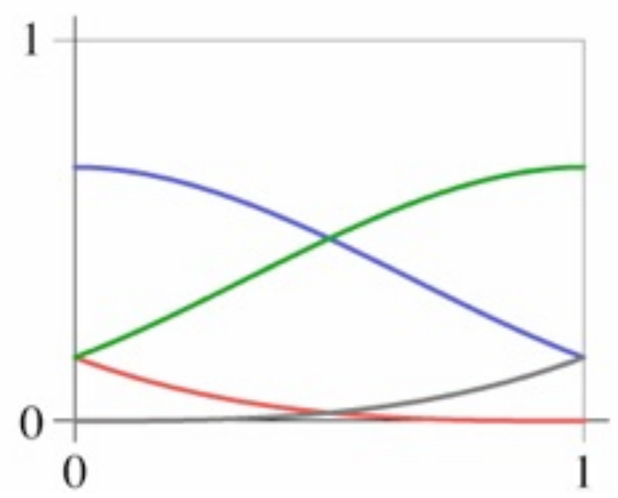
Catmull-Rom splines

- Our first example of an interpolating spline
- Like Bézier, equivalent to Hermite
  - in fact, all splines of this form are equivalent
- First example of a spline based on just a control point sequence
- Does not have convex hull property

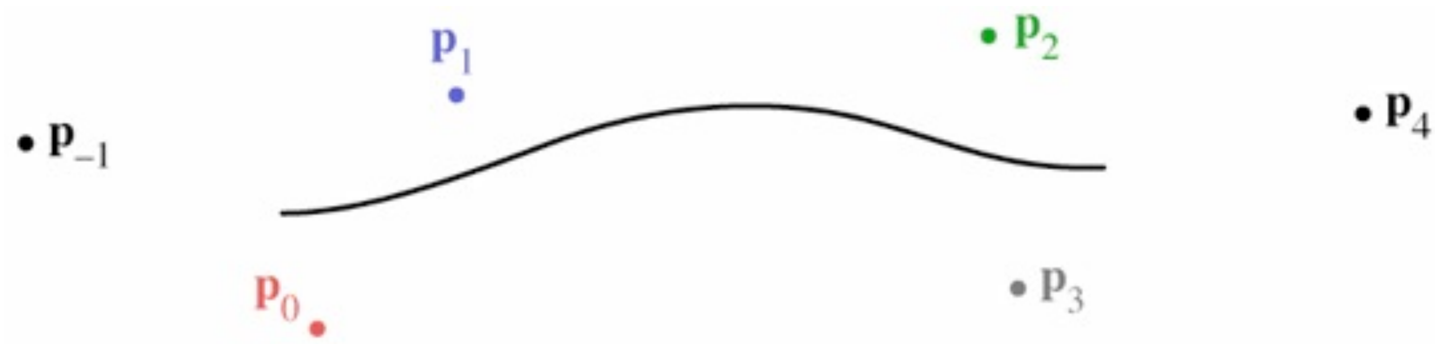
B-splines

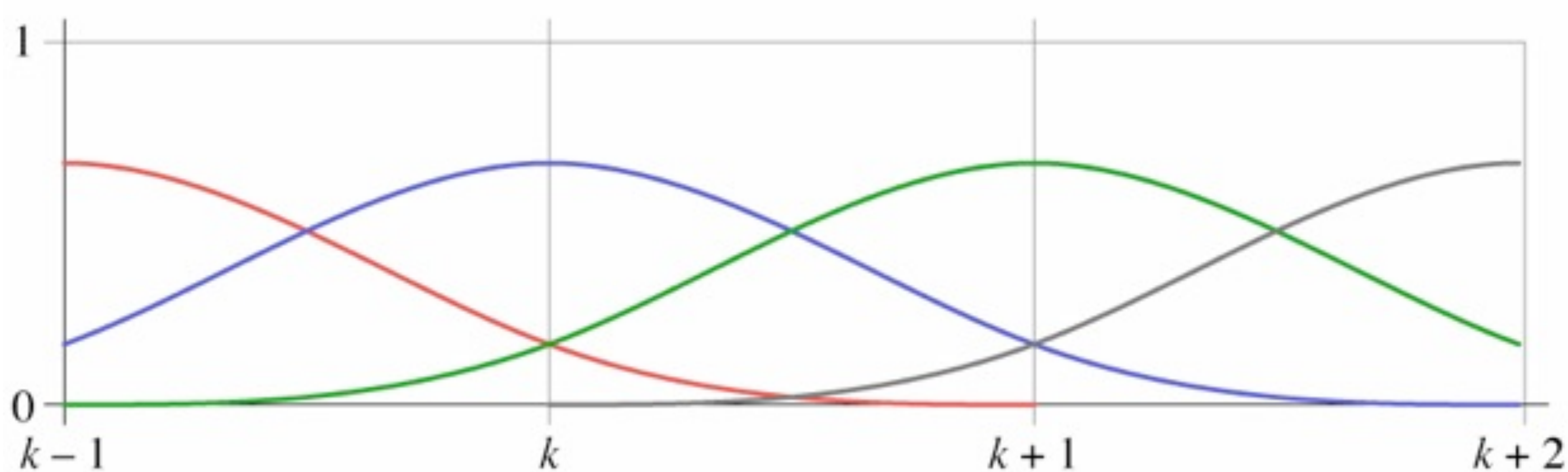
- We may want more continuity than C^1
- We may not need an interpolating spline
- B-splines are a clean, flexible way of making long splines with arbitrary order of continuity
- Various ways to think of construction
  - a simple one is convolution
  - relationship to sampling and reconstruction

Cubic B-spline basis



Cubic B-spline basis





Deriving the B-Spline

- Approached from a different tack than Hermite-style constraints
  - Want a cubic spline; therefore 4 active control points
  - Want C^2 continuity
  - Turns out that is enough to determine everything

Efficient construction of any B-spline

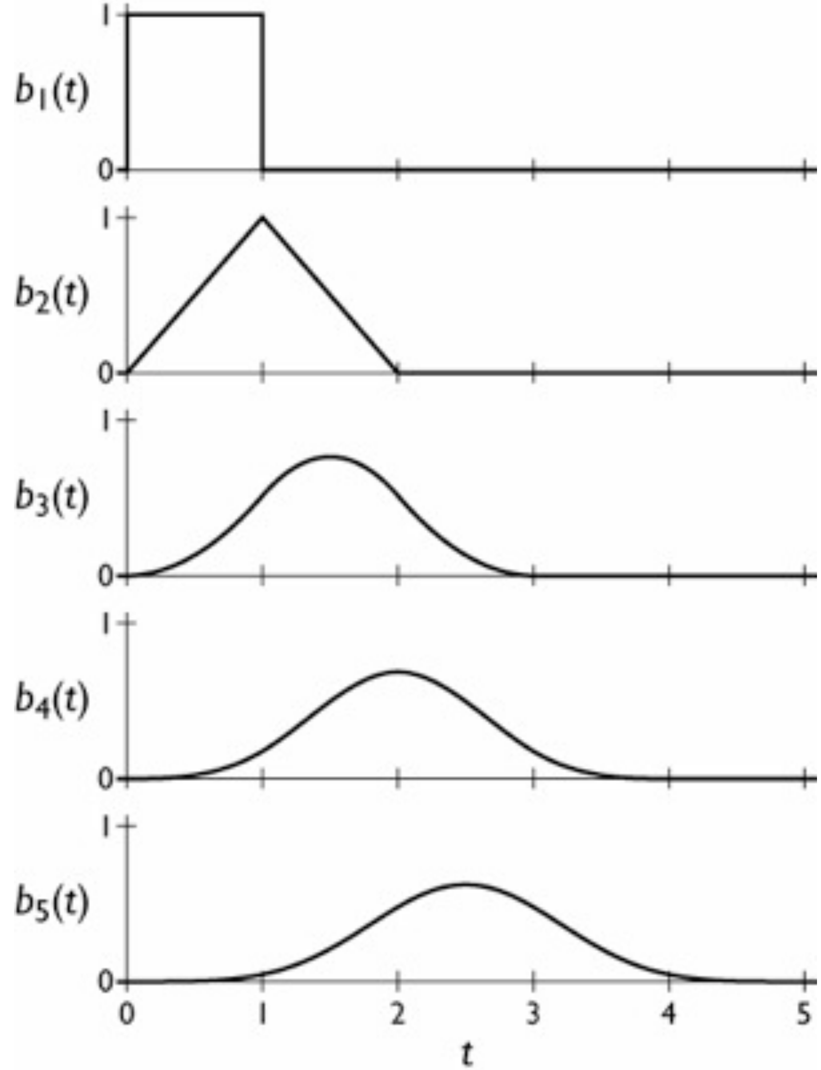
- B-splines defined for all orders
  - order *d*: degree *d* – 1
  - order *d*: *d* points contribute to value
- One definition: Cox-deBoor recurrence

$$b_1 = \begin{cases} 1 & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$b_d = \frac{t}{d-1} b_{d-1}(t) + \frac{d-t}{d-1} b_{d-1}(t-1)$$

B-spline construction, alternate view

- Recurrence
  - ramp up/down
- Convolution
  - smoothing of basis fn
  - smoothing of curve



Cubic B-spline matrix

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

Converting spline representations

- All the splines we have seen so far are equivalent
  - all represented by geometry matrices

$$\mathbf{p}_S(t) = T(t) M_S P_S$$

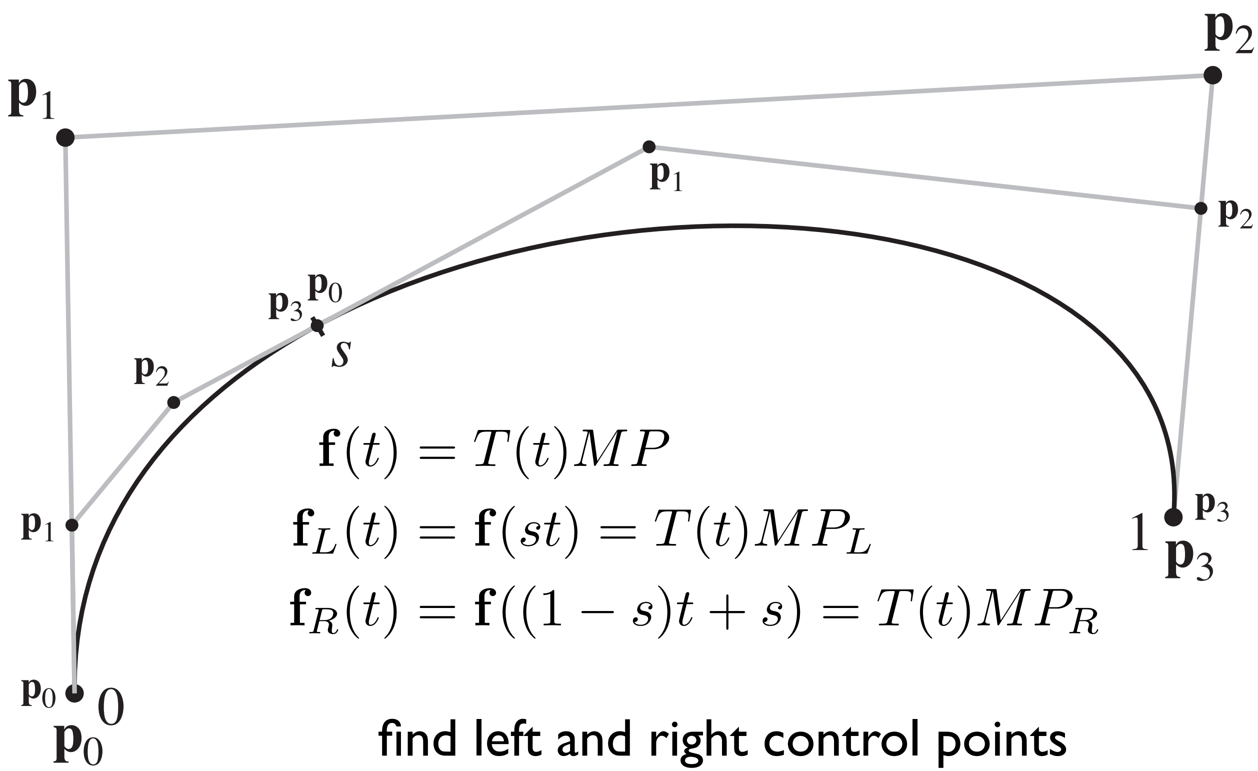
    - where *S* represents the type of spline
  - therefore the control points may be transformed from one type to another using matrix multiplication

$$P_1 = M_1^{-1} M_2 P_2$$

$$\begin{aligned}\mathbf{p}_1(t) &= T(t) M_1 (M_1^{-1} M_2 P_2) \\ &= T(t) M_2 P_2 = \mathbf{p}_2(t)\end{aligned}$$

Refinement of splines

- May want to add more control to a curve
- Can add control by splitting a segment into two



$$\mathbf{f}(t) = T(t)MP$$

$$\mathbf{f}_L(t) = \mathbf{f}(st) = T(t)MP_L$$

$$\mathbf{f}_R(t) = \mathbf{f}((1-s)t+s) = T(t)MP_R$$

find left and right control points
to make the curves match!

Refinement math

$$\begin{aligned}\mathbf{f}_L(t) &= T(st)MP = T(t)S_LMP \\ &= T(t)M(M^{-1}S_LMP) \\ &= T(t)MP_L\end{aligned}$$

$$\begin{aligned}P_L &= M^{-1}S_LMP \\ P_R &= M^{-1}S_RMP\end{aligned}$$

$$S_L = \begin{bmatrix} s^3 & & & \\ & s^2 & & \\ & & s & \\ & & & 1 \end{bmatrix}$$

$$S_R = \begin{bmatrix} s^3 & & & \\ 3s^2(1-s) & s^2 & & \\ 3s(1-s)^2 & 2s(1-s) & s & \\ (1-s)^3 & (1-s)^2 & (1-s) & 1 \end{bmatrix}$$

Other types of B-splines

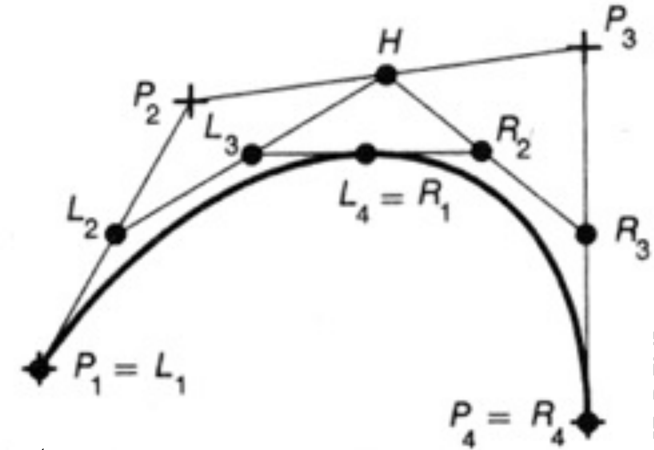
- Nonuniform B-splines
  - discontinuities not evenly spaced
  - allows control over continuity or interpolation at certain points
  - e.g. interpolate endpoints (commonly used case)
- Nonuniform Rational B-splines (NURBS)
  - ratios of nonuniform B-splines: $x(t) \,/\, w(t)$; $y(t) \,/\, w(t)$
  - key properties:
    - invariance under perspective as well as affine
    - ability to represent conic sections exactly

Evaluating splines for display

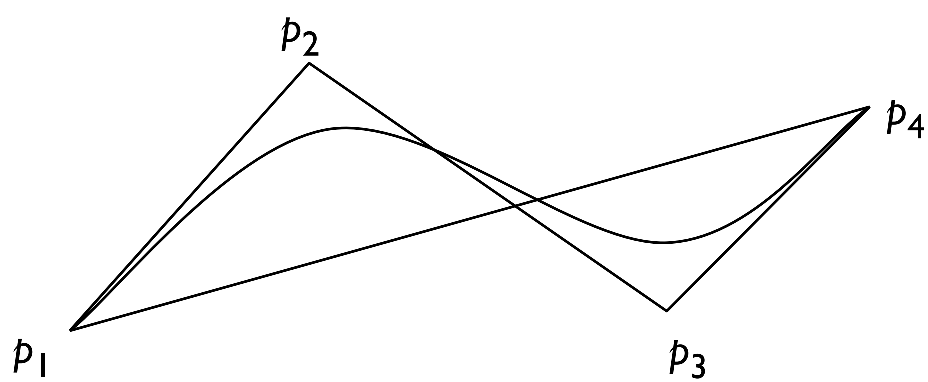
- Need to generate a list of line segments to draw
  - generate efficiently
  - use as few as possible
  - guarantee approximation accuracy
- Approaches
  - recursive subdivision (easy to do adaptively)
  - uniform sampling (easy to do efficiently)

Evaluating by subdivision

- Recursively split spline
  - stop when polygon is within epsilon of curve
- Termination criteria
  - distance between control points
  - distance of control points from line

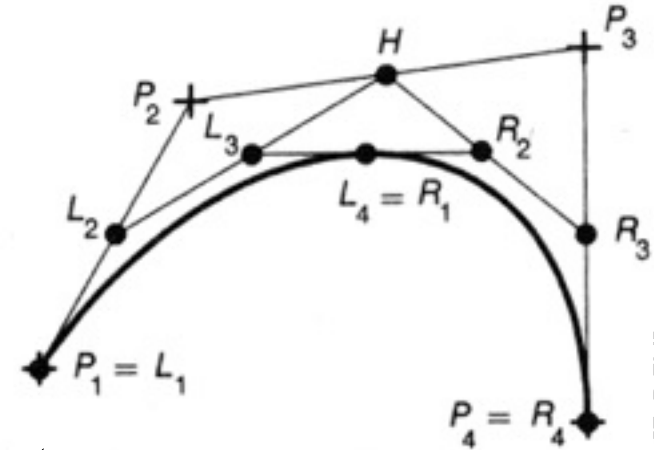


[FvDFH]

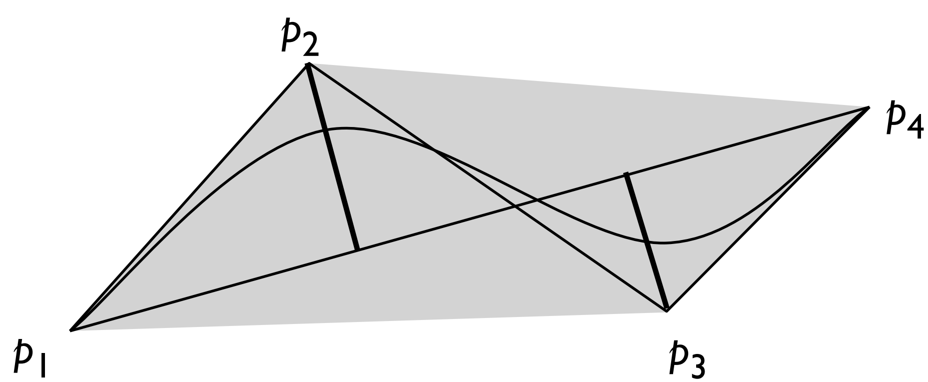


Evaluating by subdivision

- Recursively split spline
  - stop when polygon is within epsilon of curve
- Termination criteria
  - distance between control points
  - distance of control points from line



[FvDFH]



Evaluating with uniform spacing

- Forward differencing
  - efficiently generate points for uniformly spaced *t* values
  - evaluate polynomials using repeated differences

