05. Splines & Subdivision Curves

Dr. Hamish Carr

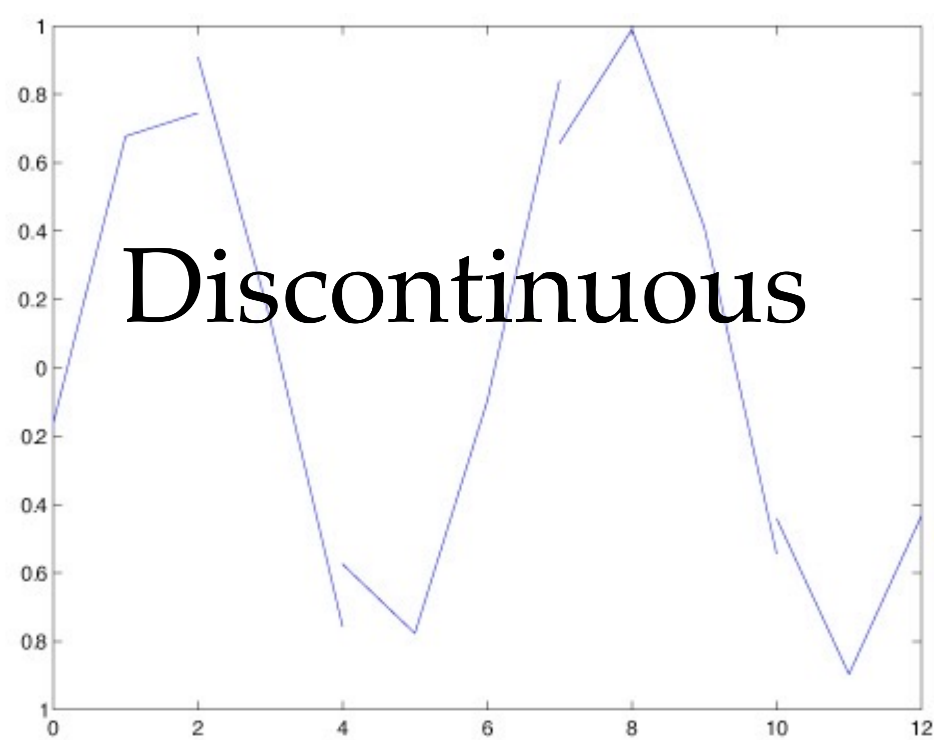
UNIVERSITY OF LEEDS

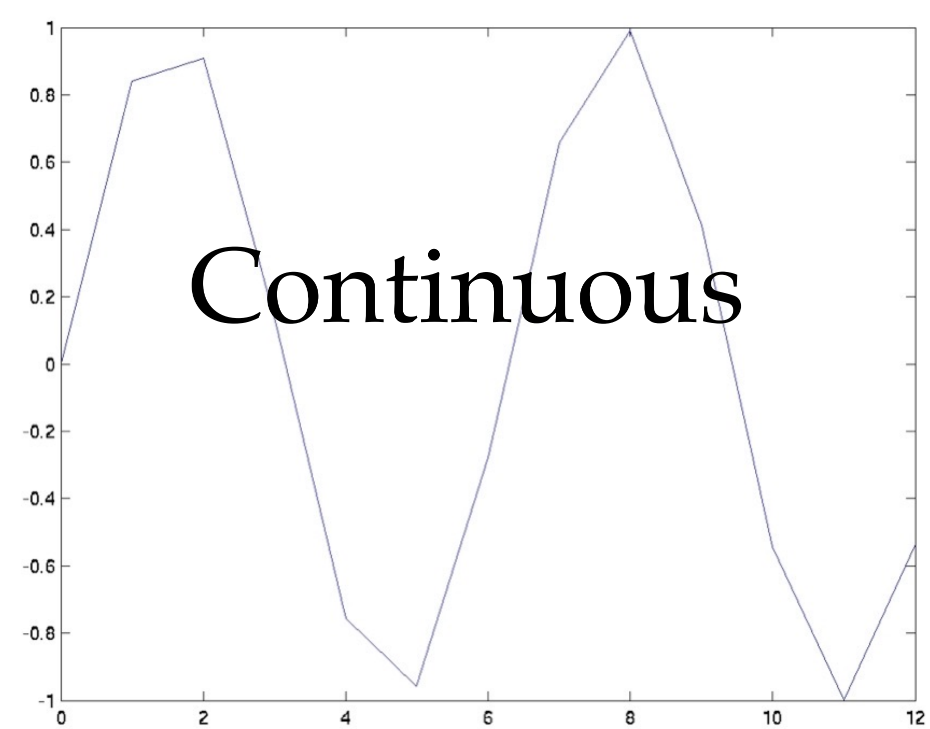
Continuity

- A continuous function *f(x)* satisfies:

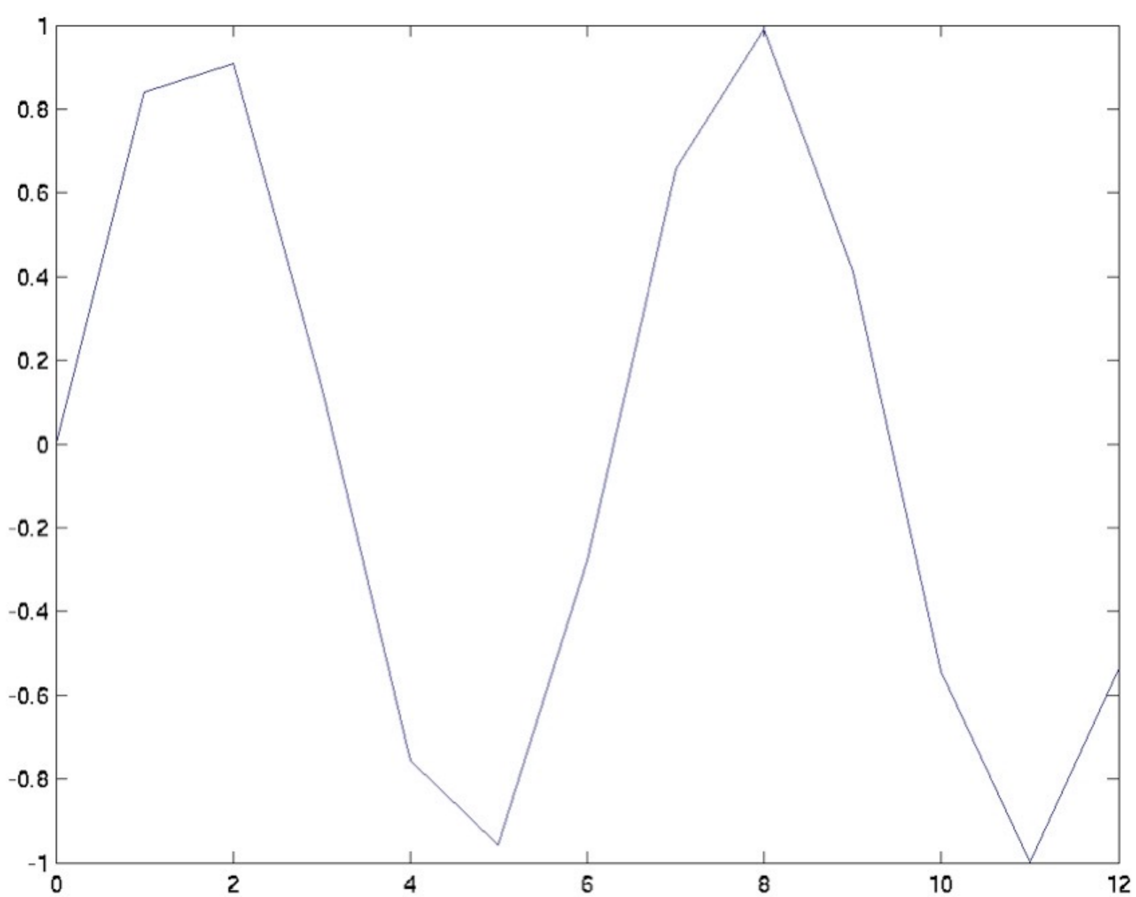
$$\lim_{x \to a^-} f(x) = f(a) = \lim_{x \to a^+} f(x)$$

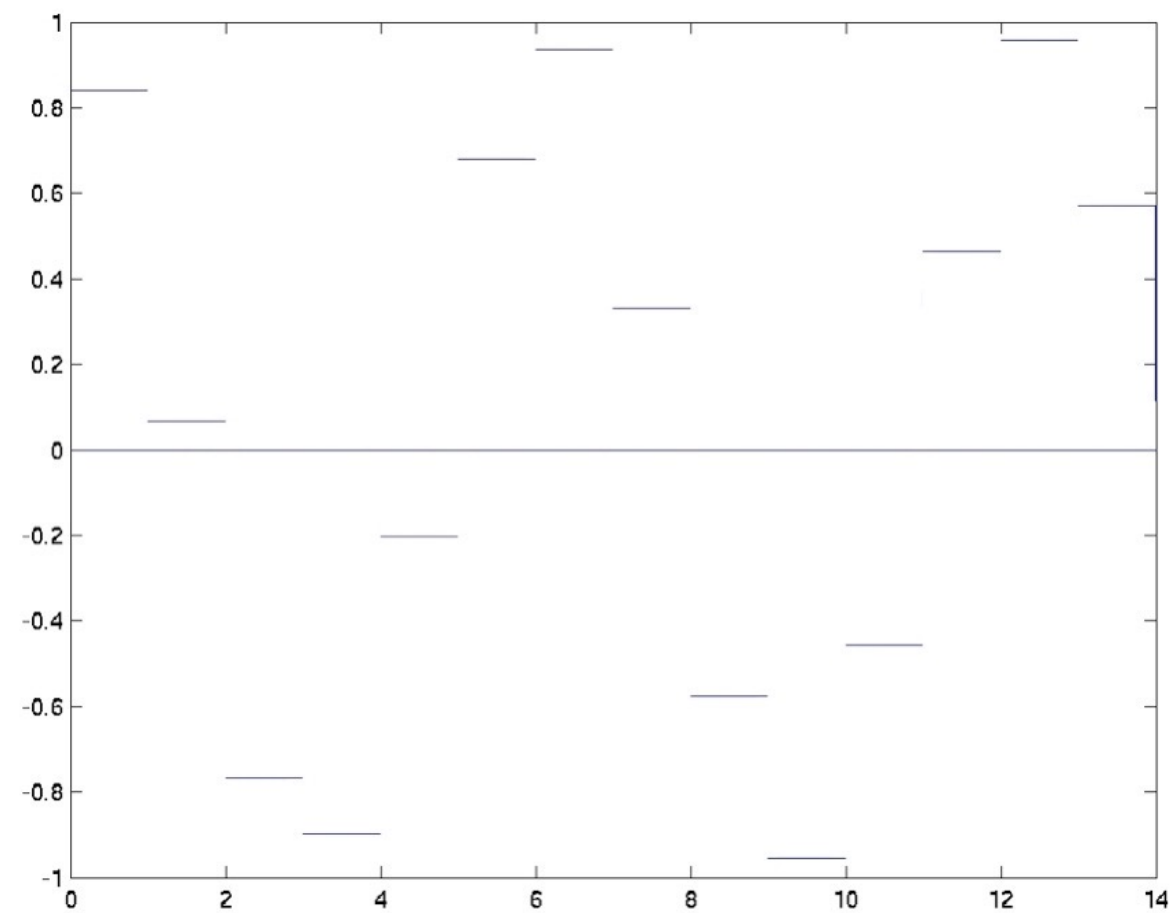
  - Also called C^0 continuous





Continuous ≠ Smooth





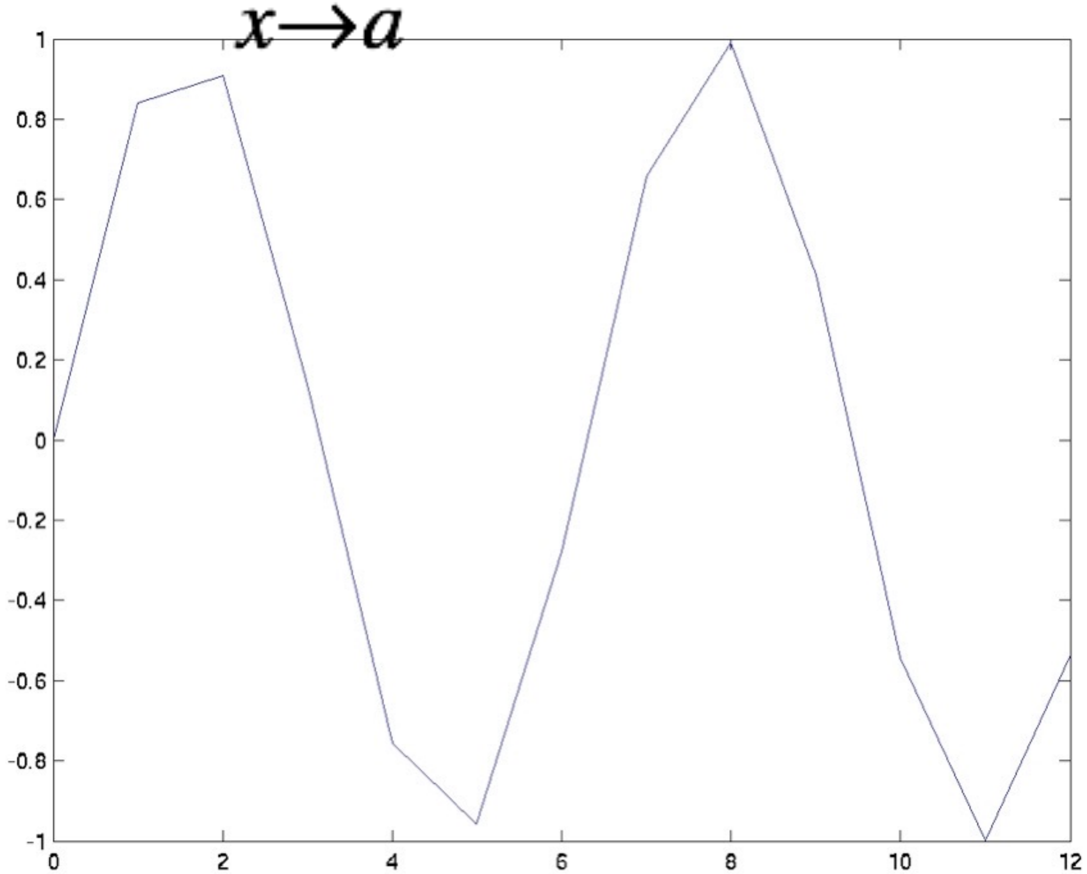
Not *smooth*
Why not?

Slope (derivative)
slope is discontinuous

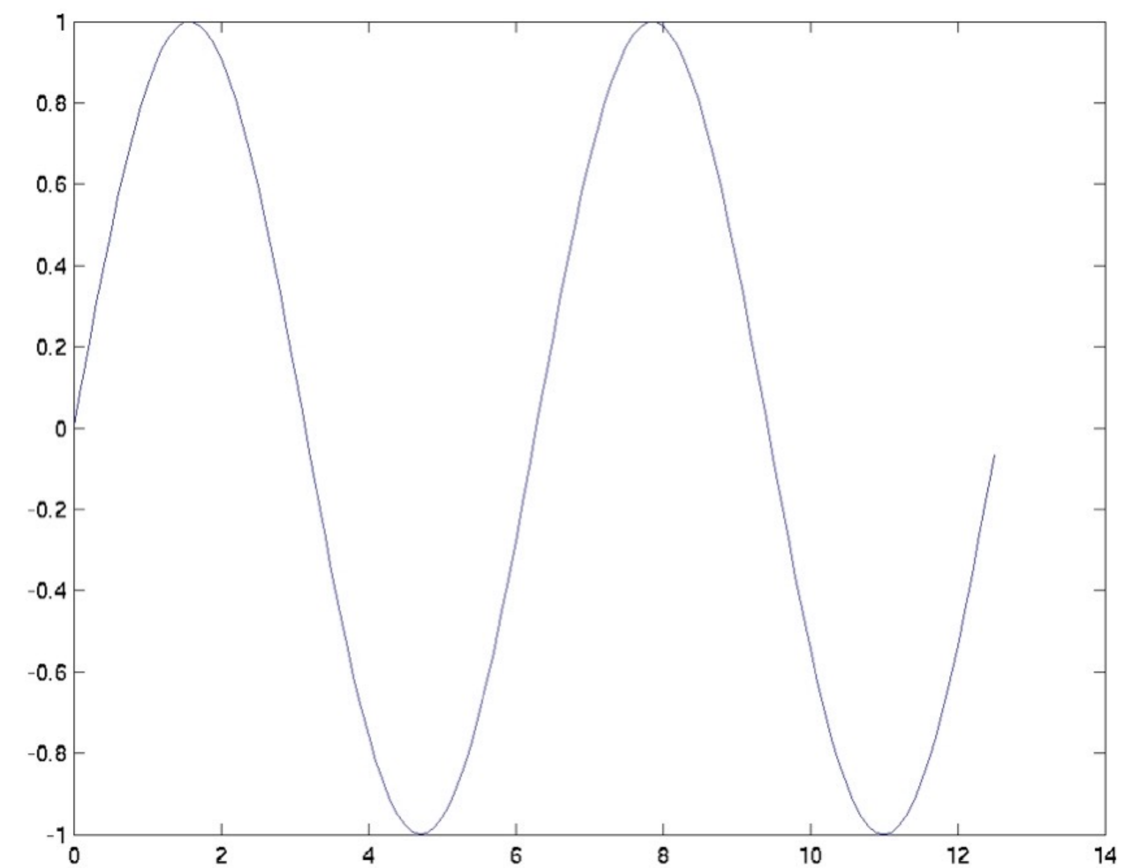
C^n Continuity

- A function *f(x)* is *C^n continuous* if:

$$\lim_{x \to a^-} f^{(n)}(x) = f^{(n)}(a) = \lim{}^{(n)} f(x)$$



C^0 Continuous



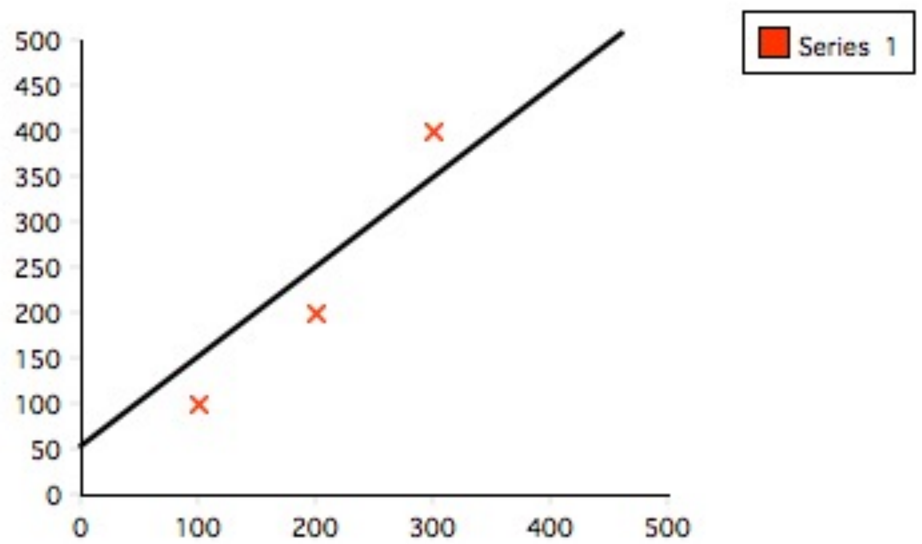
C^1 Continuous

actually C^∞

Desiderata

- Smooth curves -> at least C^1 continuity
- Mathematically simple -> polynomials
- Closed form intersections -> polynomials
- Computationally efficient -> low-order
- Interpolating -> pass through points
- Local control -> artists can use them
- Bounded interpolation -> ditto

Least Squares



- Find *one* line that fits a set of points

$$R = \sum_{i=1}^{N} d_i^2$$
$$= \sum_{i=1}^{N} \left[y_i - \left(a + bx_i \right) \right]^2$$

- Minimise this: solve for a, b

- Non-interpolating
- Only gives you a line

Lagrange Polynomials

- Interpolating (+)
- Polynomial (+)
- High-order (-)
  - Inefficient (-)
  - No closed form (-)
- No local control

$$
\begin{aligned}
\alpha_{ij}(x) &= \frac{x - x_j}{x_i - x_j} \\
&= \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}
\end{aligned}
$$

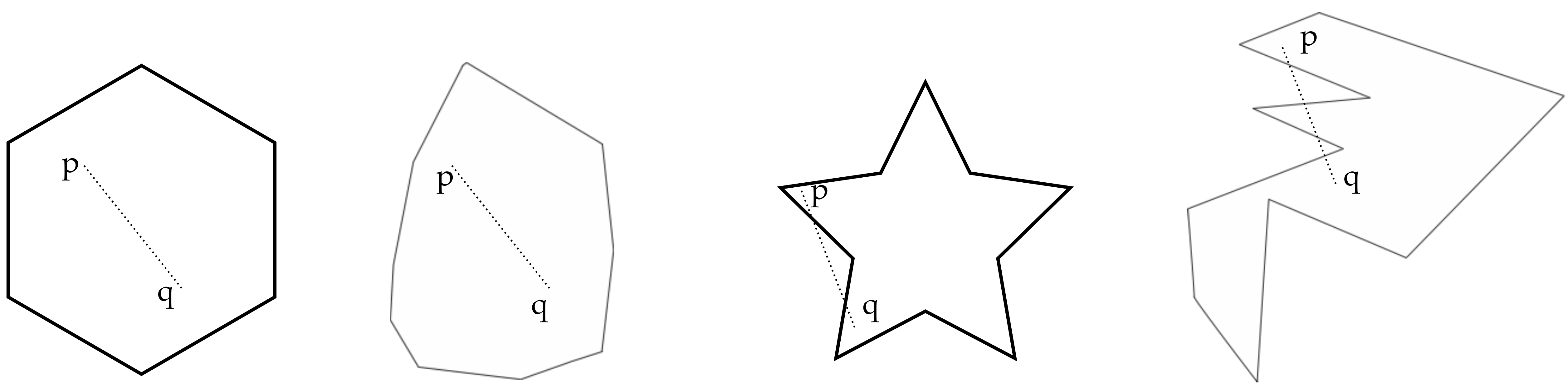
$$
\begin{aligned}
l_j(x) &= \prod_{\substack{i=1 \\ i \neq j}}^{N} \alpha_{ij}(x) \\
&= \prod_{\substack{i=1 \\ i \neq j}}^{N} \frac{x - x_j}{x_i - x_j}
\end{aligned}
$$

$$
\begin{aligned}
f(x) &= \sum_{j=1}^{N} l_j(x) y_j \\
&= \sum_{j=1}^{N} \prod_{\substack{i=1 \\ i \neq j}}^{N} \frac{x - x_j}{x_i - x_j} y_j
\end{aligned}
$$

Lagrange Example

Convex Polygons

- In a convex polygon,
  - if p,q are in the polygon
  - then so is the line pq

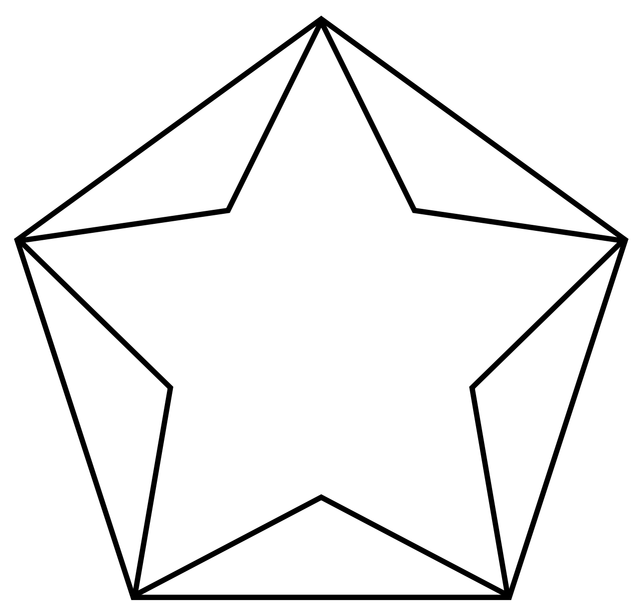


Convex

Concave

Convex Hull

- For a set of points
  - the convex hull is the
    - smallest convex polygon
    - that contains all the points



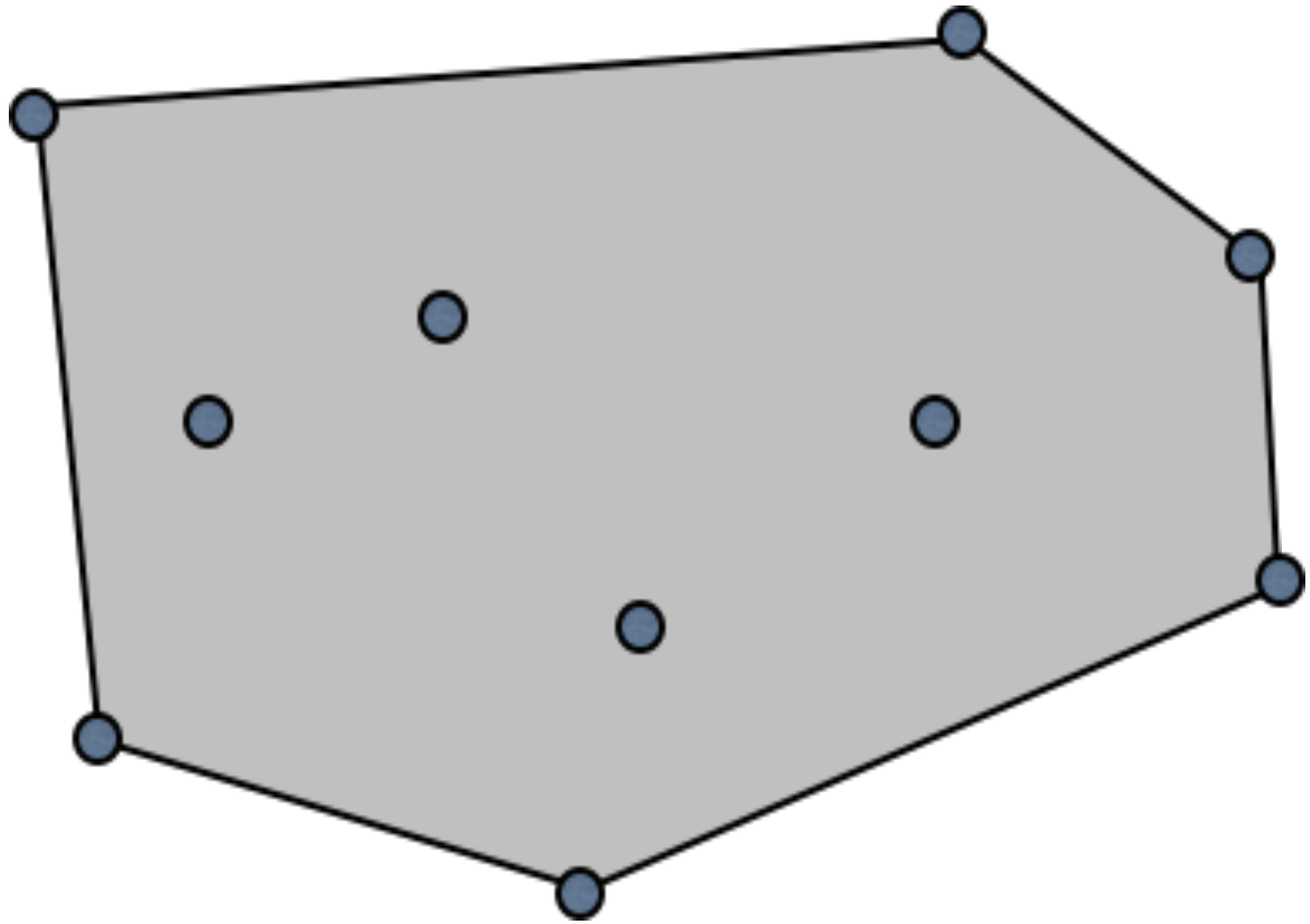
Formal Definition

- The *convex hull* of $\{p_i = (x_i, y_i)\}$ is the set:

$$\left\{ p = \sum_{i=1}^{n} \alpha_i p_i : \alpha_i \in [0,1], \sum_{i=1}^{n} \alpha_i = 1 \right\}$$

- Extension of barycentric interpolation to a set
- Not guaranteed to be linear
- But provides guarantees on behaviour
  - e.g. keep curve in convex hull of point set

Convex Hull Example



- *All* linear combinations of points
- Algorithms later ...

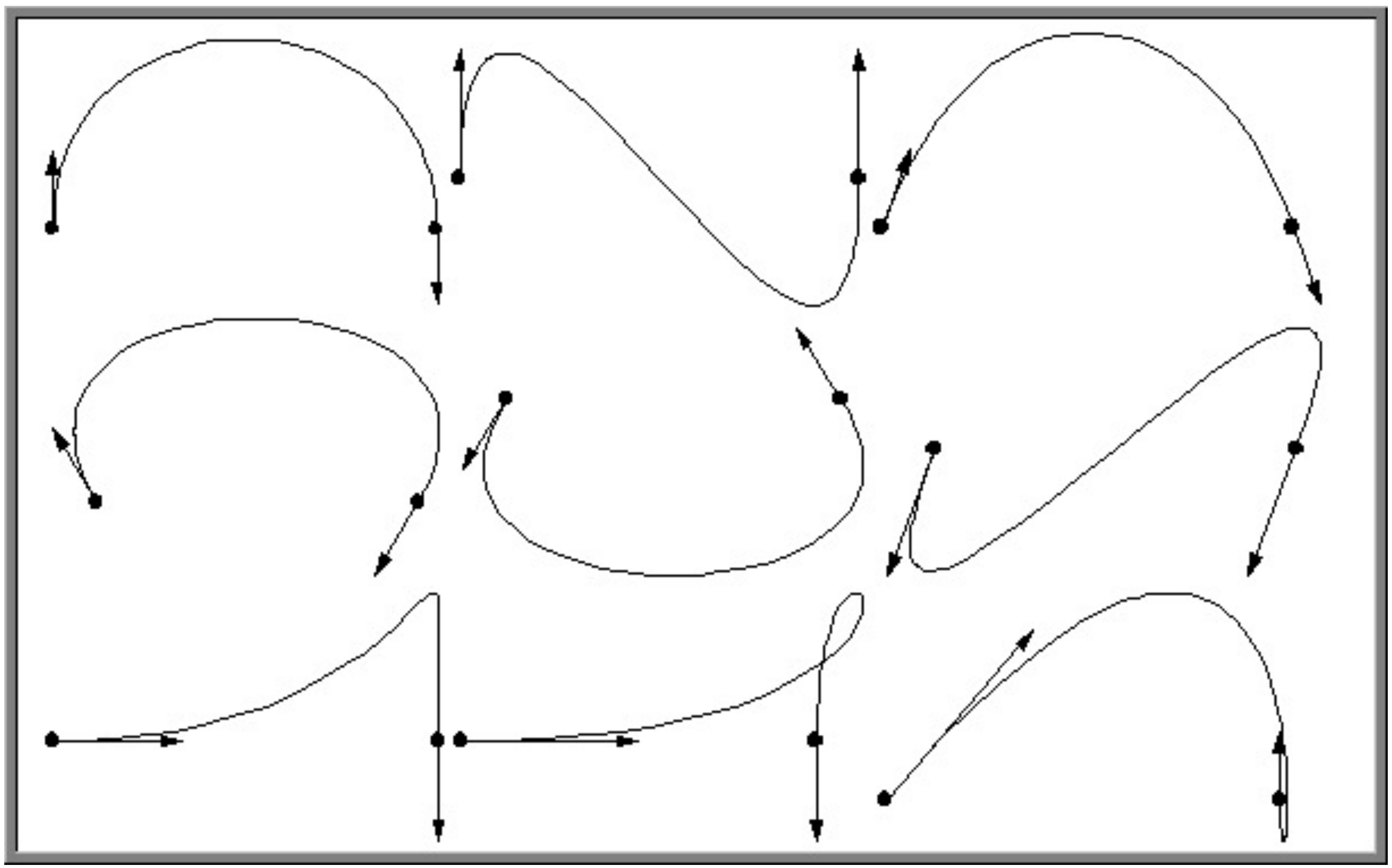
Interpolating Conditions

- Extra interpolating points are difficult
- So add:
  - interpolating vectors (Hermites)
  - non-interpolating points (Béziers / Splines)
- They're all equivalent anyway
  - And turn out to require cubic polynomials

Hermite Conditions

- We want a curve that passes through 2 points
  - $f(0) = p_0$
  - $f(1) = p_1$
- And has known *direction* vectors
  - $f'(0) = \vec{v}_0$
  - $f'(1) = \vec{v}_1$

Hermite Examples



©T. Munzner, UBC

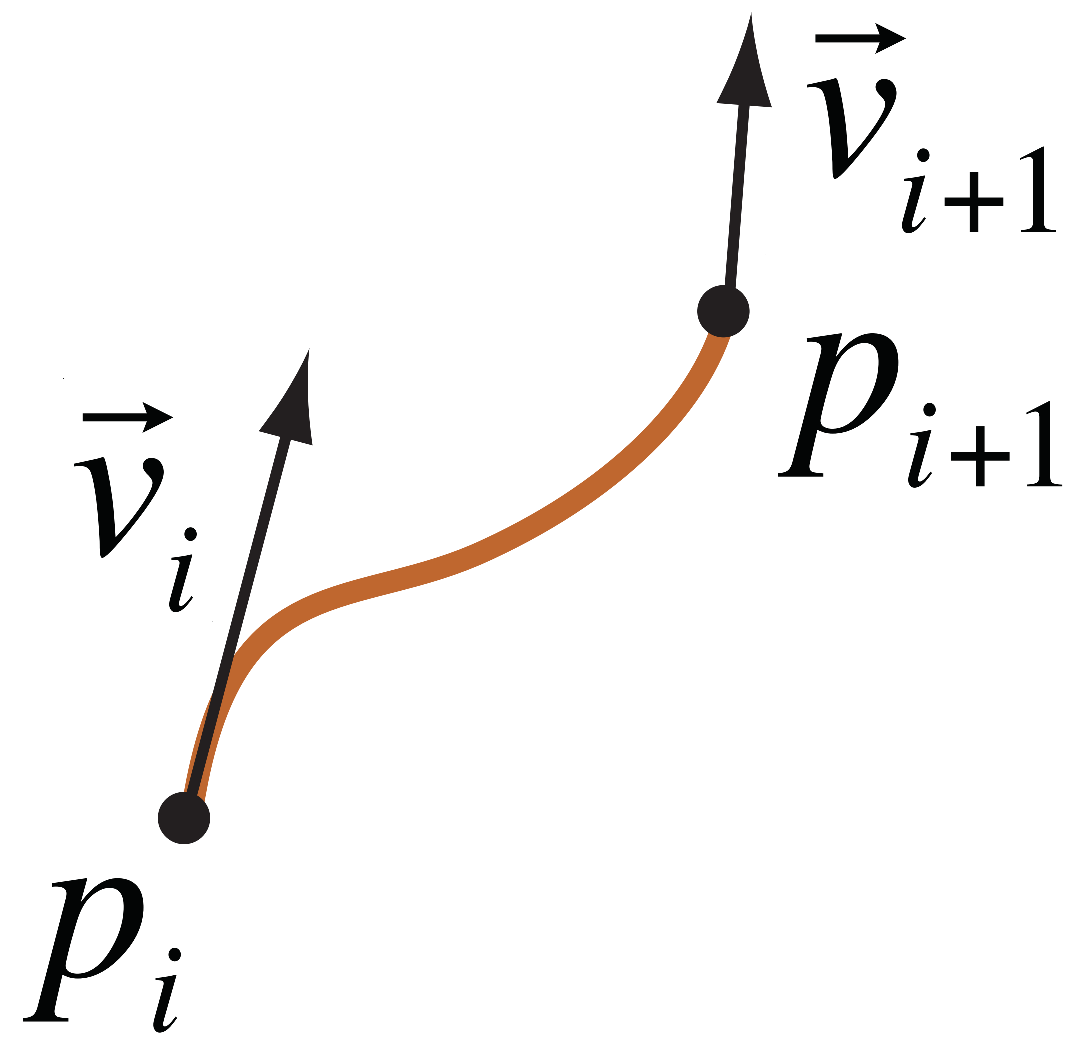
Hermite Example

Piecewise Hermites

- We want to link them together
- As we use line segments to build polygons
- So we will assume we have a sequence
    - $\{f_0, f_1, \ldots, f_n\}$
- We will assume points/vectors are repeated

Cubic Hermite Curves

- Given by:
  - $f_i(0) = p_{\mathrm{i}}$
  - $f_i(1) = p_{\mathrm{i+1}}$
  - $f_i'(0) = \vec{v}_i$
  - $f_i'(1) = \vec{v}_{i+1}$



Development

$$
\begin{aligned}
f_i(t) &= a_i t^3 + b_i t^2 + c_i t + d_i \\
\\
p_i &= f_i(0) \\
&= a_i 0^3 + b_i 0^2 + c_i 0 + d_i \\
&= d_i \\
\\
f_i'(t) &= 3a_i t^2 + 2b_i t + c_i \\
\\
\vec{v}_i &= 3a_i 0^2 + 2b_i 0 + c_i \\
&= c_i \\
p_{i+1} &= f_i(1) \\
&= a_i 1^3 + b_i 1^2 + c_i 1 + d_i \\
&= a_i + b_i + c_i + d_i \\
&= a_i + b_i + \vec{v}_i + p_i \\
p_{i+1} - p_i - \vec{v}_i &= a_i + b_i \\
\vec{w}_i - \vec{v}_i &= a_i + b_i
\end{aligned}
$$

$$
\begin{aligned}
\vec{v}_{i+1} &= 3a_i 1^2 + 2b_i 1 + c_i \\
&= 3a_i + 2b_i + \vec{v}_i \\
\vec{v}_{i+1} - \vec{v}_i &= 3a_i + 2b_i \\
\\
\vec{v}_{i+1} &= 3a_i 1^2 + 2b_i 1 + c_i \\
&= 3a_i + 2b_i + \vec{v}_i \\
\vec{v}_{i+1} - \vec{v}_i &= 3a_i + 2b_i
\end{aligned}
$$

$$
\begin{aligned}
b_i &= 3p_{i+1} - 3p_i - 2\vec{v}_i - \vec{v}_{i+1} \\
a_i + b_i &= p_i + 1 - p_i - \vec{v}_i \\
a_i + 3p_{i+1} - 3p_i - 2\vec{v}_i - \vec{v}_{i+1} &= p_i + 1 - p_i - \vec{v}_i \\
a_i &= 2p_i - 2p_{i+1} + \vec{v}_i + \vec{v}_{i+1}
\end{aligned}
$$

Finally!!

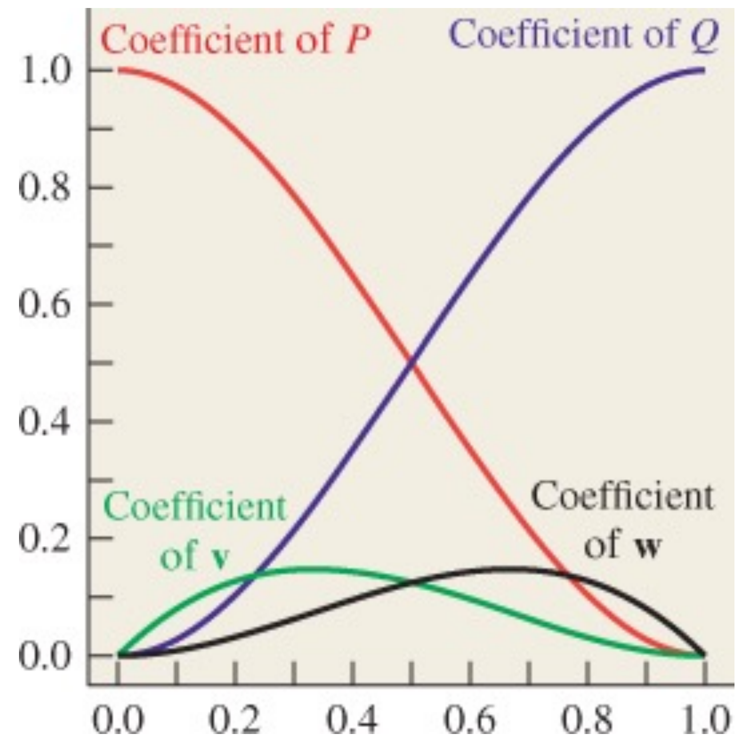
$$f_i(t) \quad = \quad \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_i \\ p_{i+1} \\ \vec{v}_i \\ \vec{v}_{i+1} \end{bmatrix}$$

Basis matrix M

Geometry matrix G

- Now we have something workable
  - interpolating, but uses vectors

Basis Functions

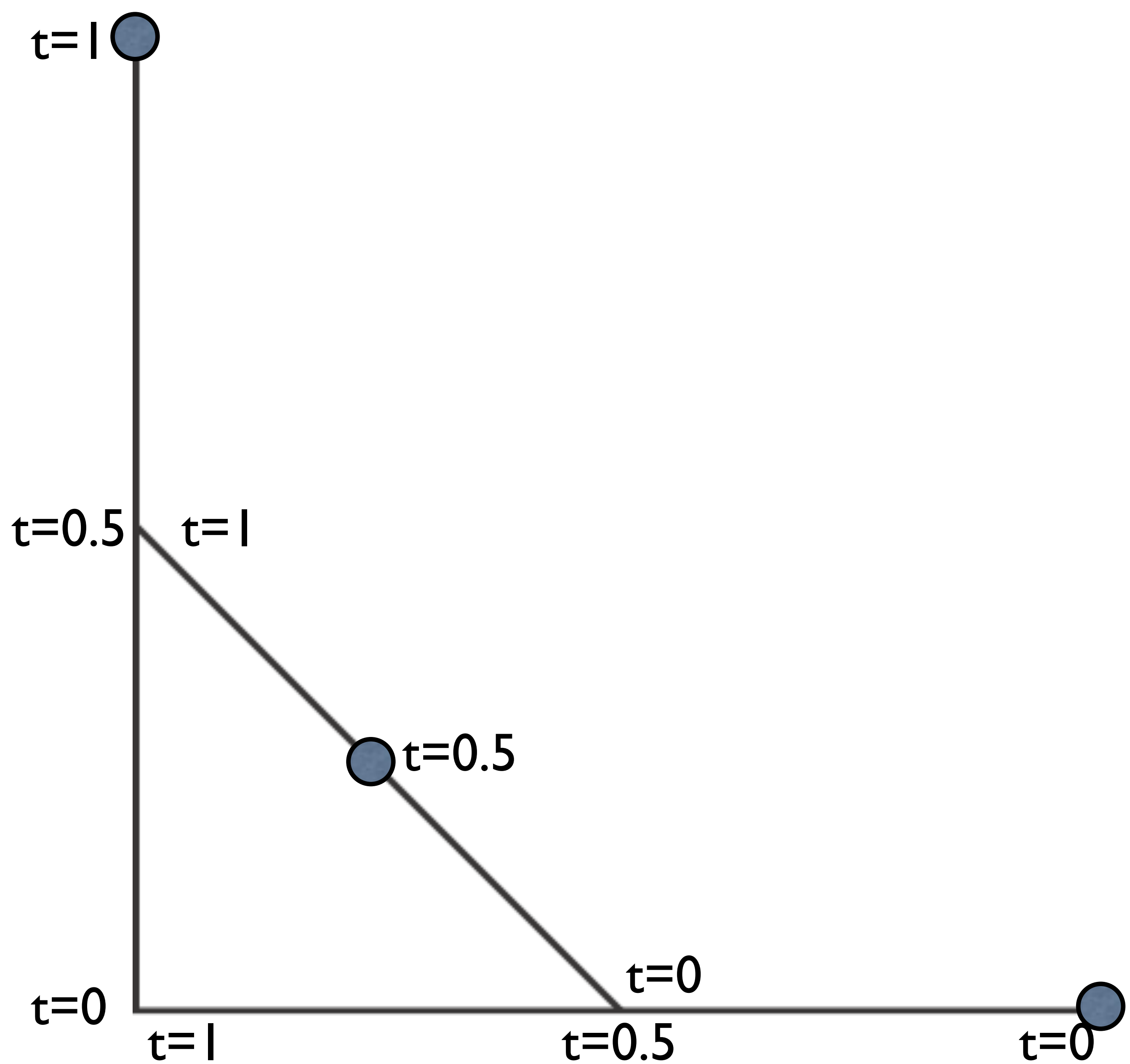


- We can plot the *weight* of each point/vector
  - as a function of parameter t

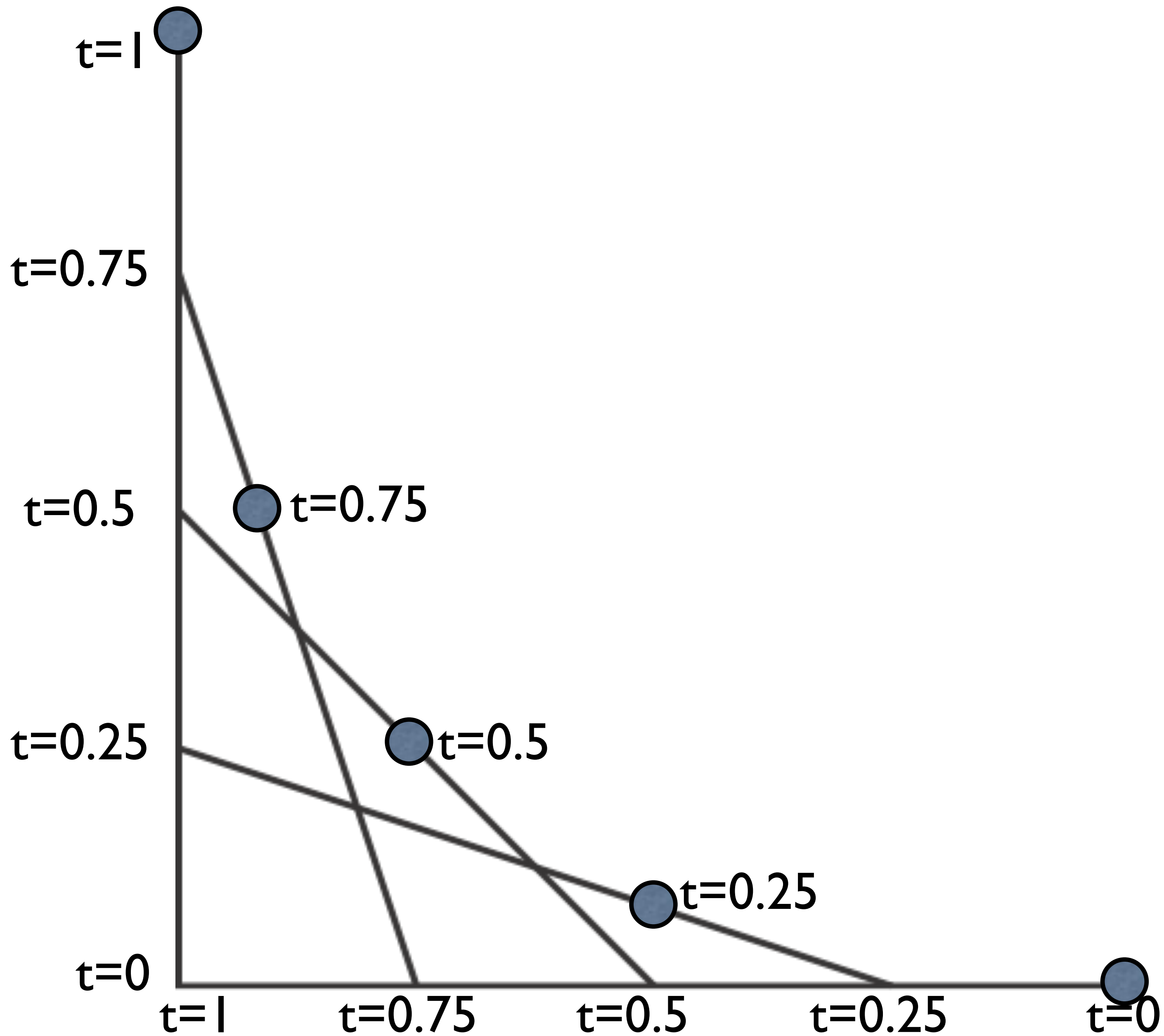
Bézier Curves

- Arbitrary polynomial degree
- Cubic Béziers are equivalent to Hermites
  - And can be converted to/from them
- But easier to implement
- And easier to generalise

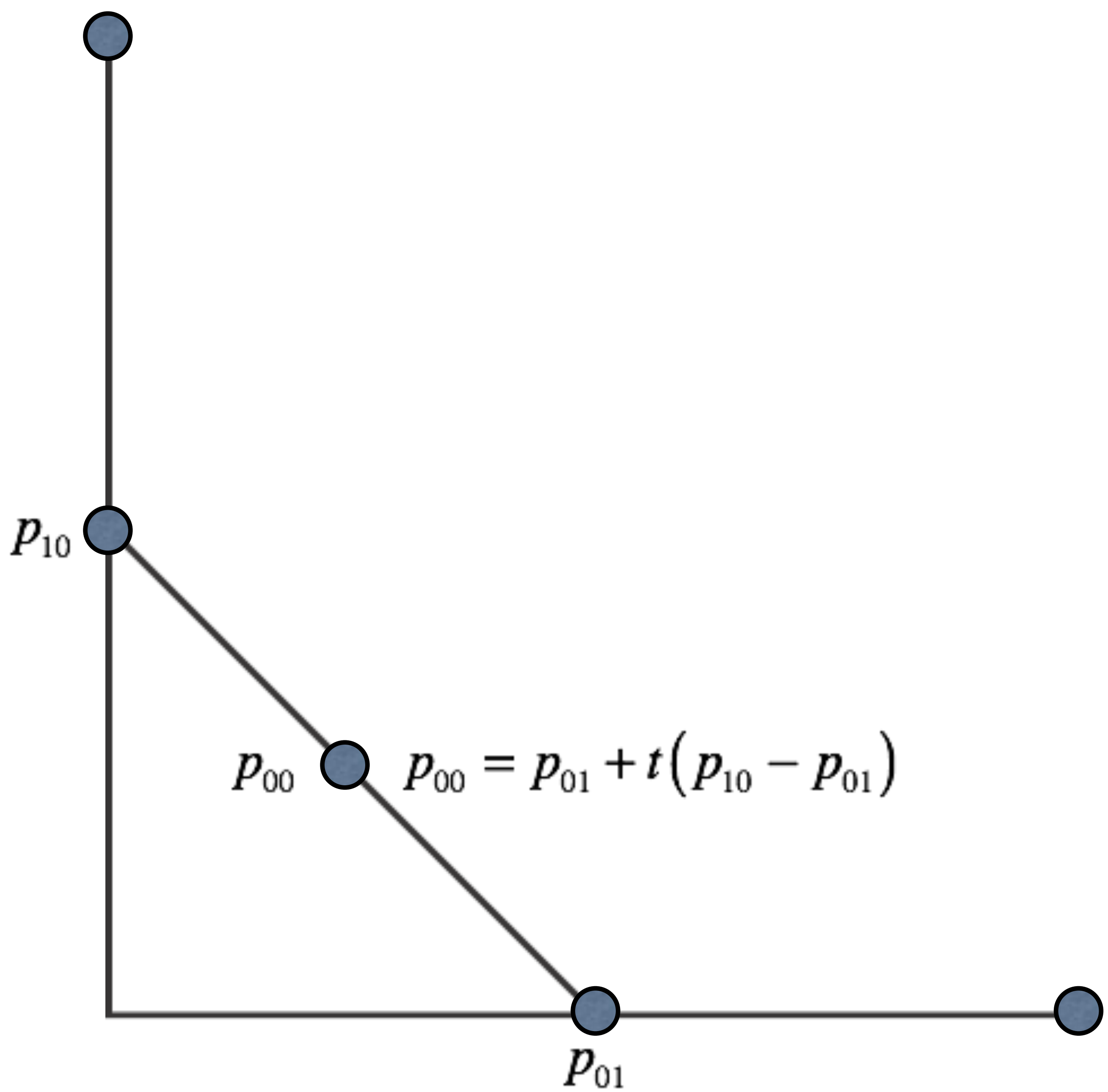
Parametrization



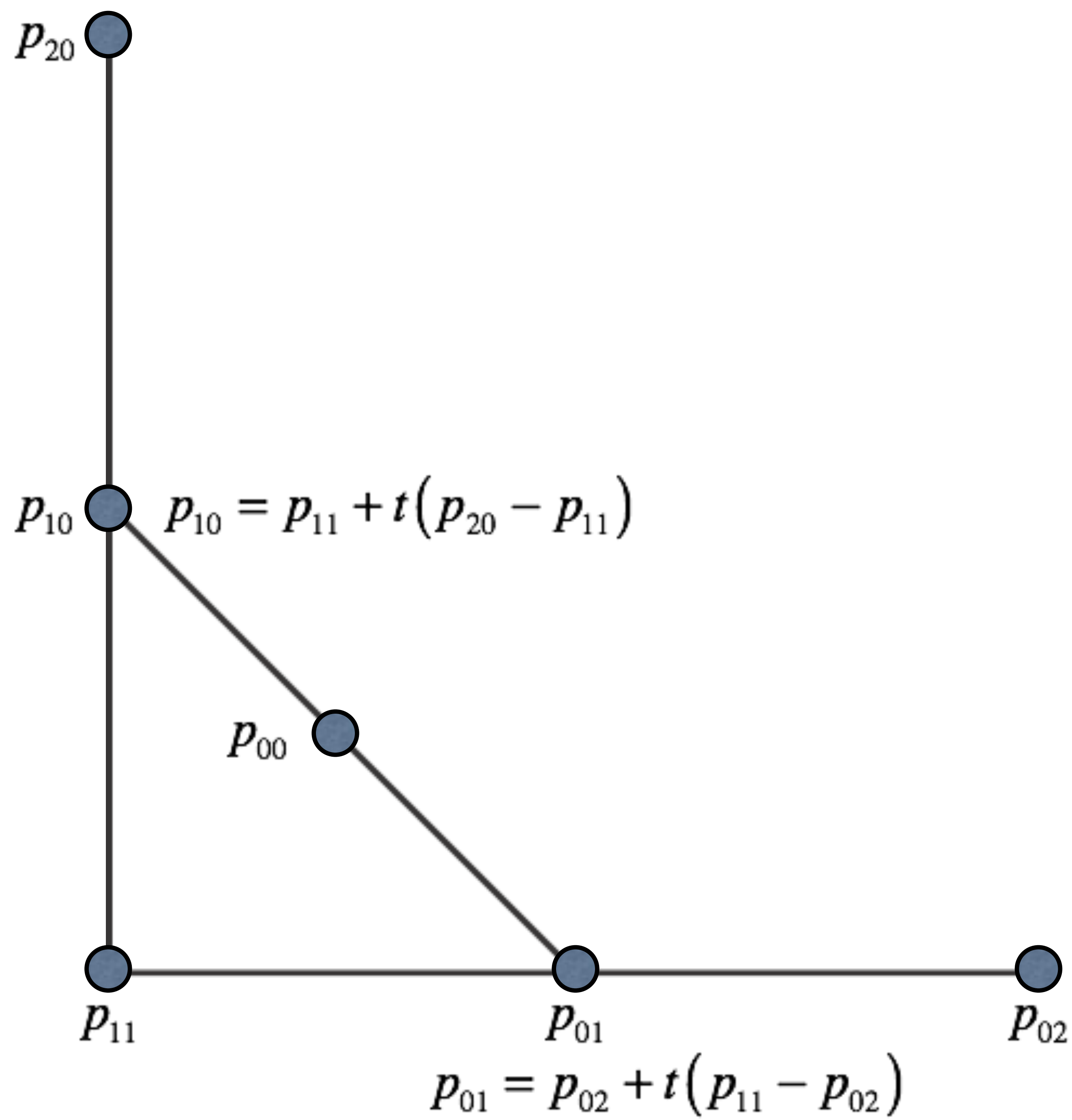
Parametrization



Development



Development



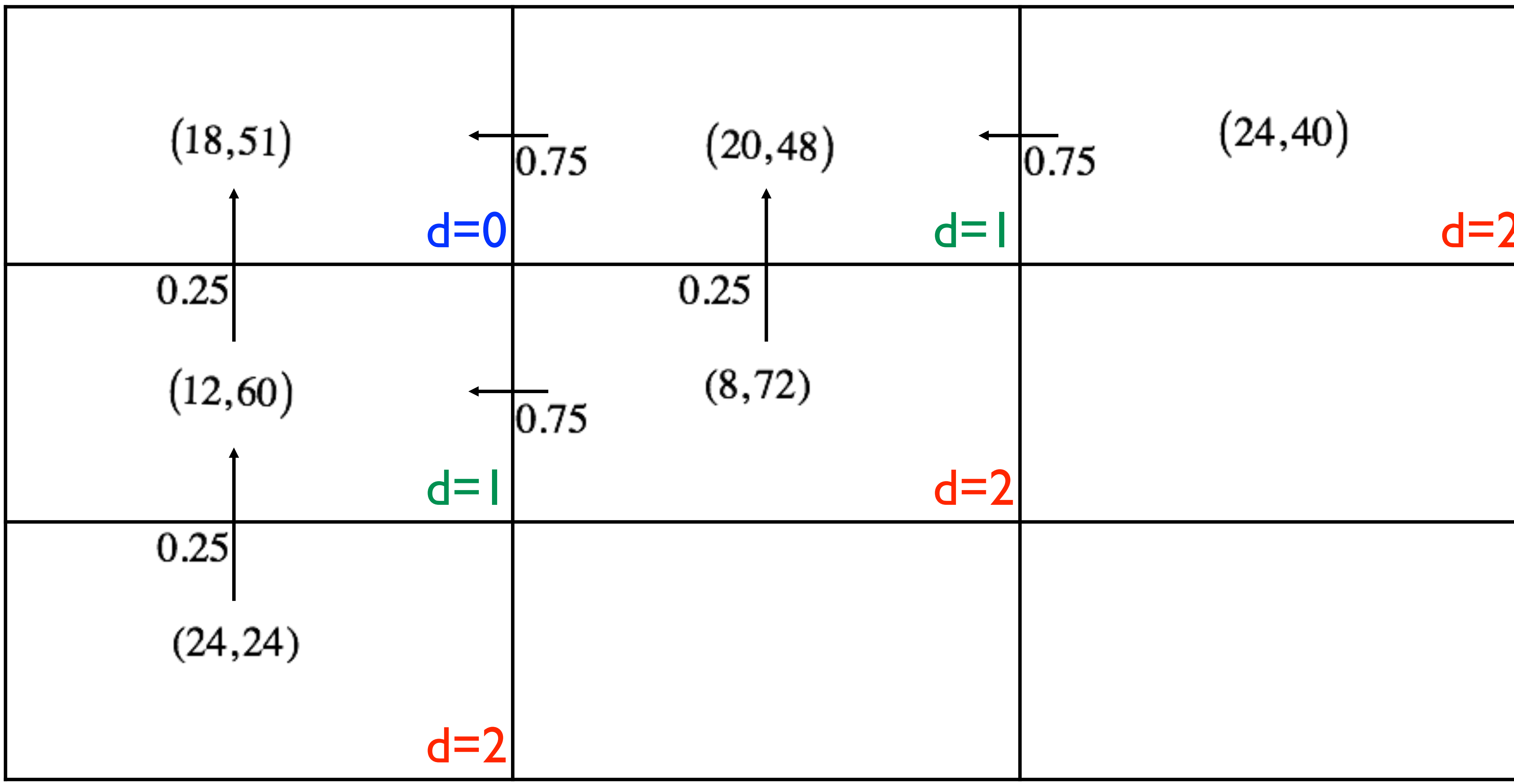
Algebra

$$
\begin{aligned}
p_{10} &= p_{11} + t\left(p_{20} - p_{11}\right) = \left(1-t\right)p_{11} + t\left(p_{20}\right) \\
p_{01} &= p_{02} + t\left(p_{11} - p_{02}\right) = \left(1-t\right)p_{02} + t\left(p_{11}\right) \\
p_{00} &= p_{01} + t\left(p_{10} - p_{01}\right) = \left(1-t\right)p_{01} + t\left(p_{10}\right) \\
&= \left(1-t\right)\left(\left(1-t\right)p_{02} + t\left(p_{11}\right)\right) + t\left(\left(1-t\right)p_{11} + t\left(p_{20}\right)\right) \\
&= p_{02} - 2p_{02}t + p_{02}t^2 + p_{11}t - p_{11}t^2 + p_{11}t - p_{11}t^2 + p_{20}t^2 \\
&= \left(p_{02} - 2p_{11} + p_{20}\right)t^2 \\
&\quad + \left(-2p_{02} + 2p_{11}\right)t \\
&\quad + p_{02} \\
&= \begin{bmatrix} p_{02} & p_{11} & p_{20} \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^2 \\ t \\ 1 \end{bmatrix}
\end{aligned}
$$

Table method

$p_{00} = (1-t)p_{01} + t(p_{10}) \leftarrow$ d=0	$1-t$ $p_{01} = (1-t)p_{02} + t(p_{11}) \leftarrow$ d=1	$1-t$ p_{02} d=2
t $\uparrow$ $p_{10} = (1-t)p_{11} + t(p_{20}) \leftarrow$ d=1	t $\uparrow$ $1-t$ p_{11} d=2	
t $\uparrow$ p_{20} d=2		

Table method



In general

- Compute diagonals in descending order d
- Each entry is found by:

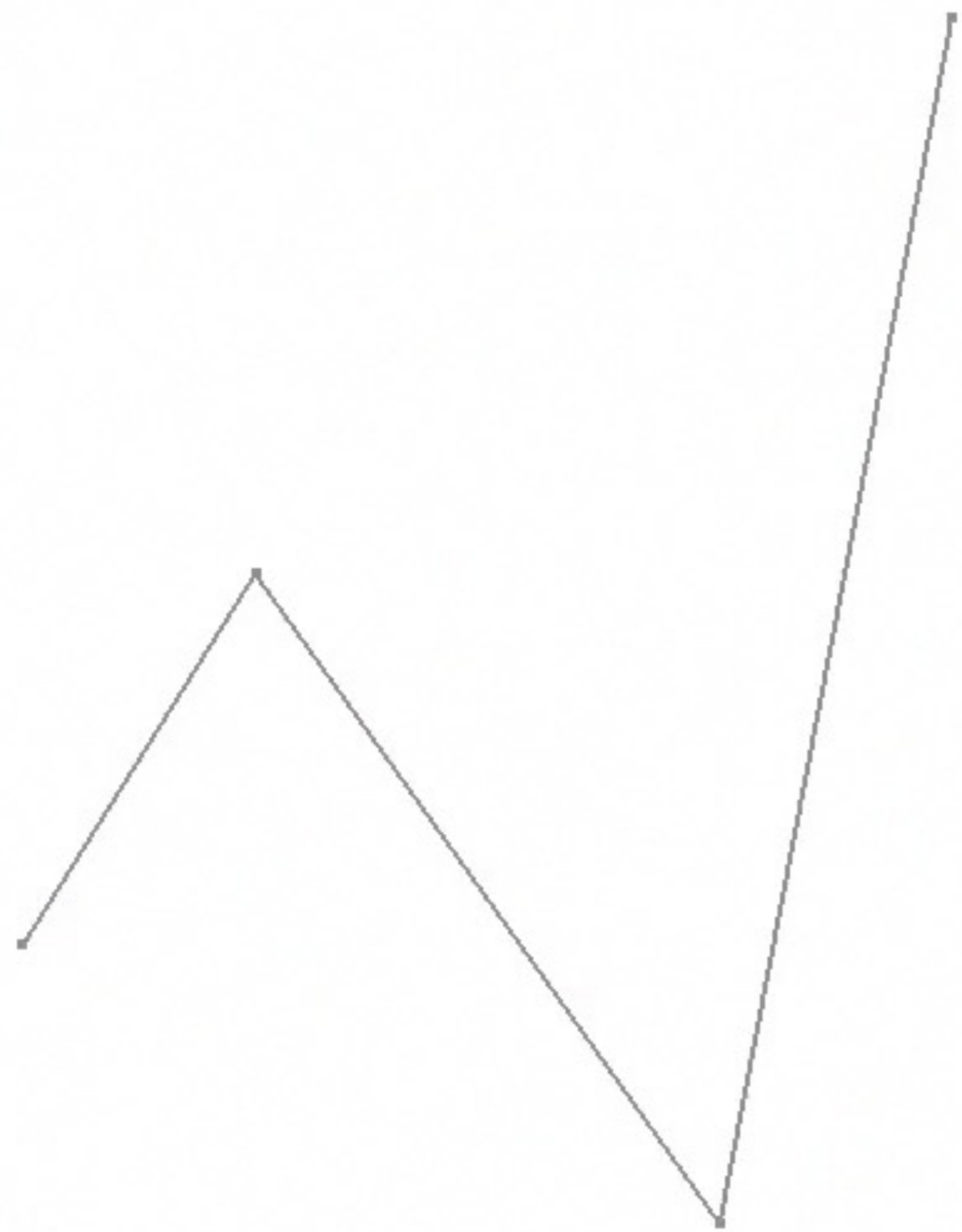
$$p_{i,j} = (1-t)p_{i,j+1} + tp_{i+1,j} \ : i + j = d$$

- We stop when we reach p_{00}
- And draw it
- Repeat for different values of t

de Casteljau Algorithm

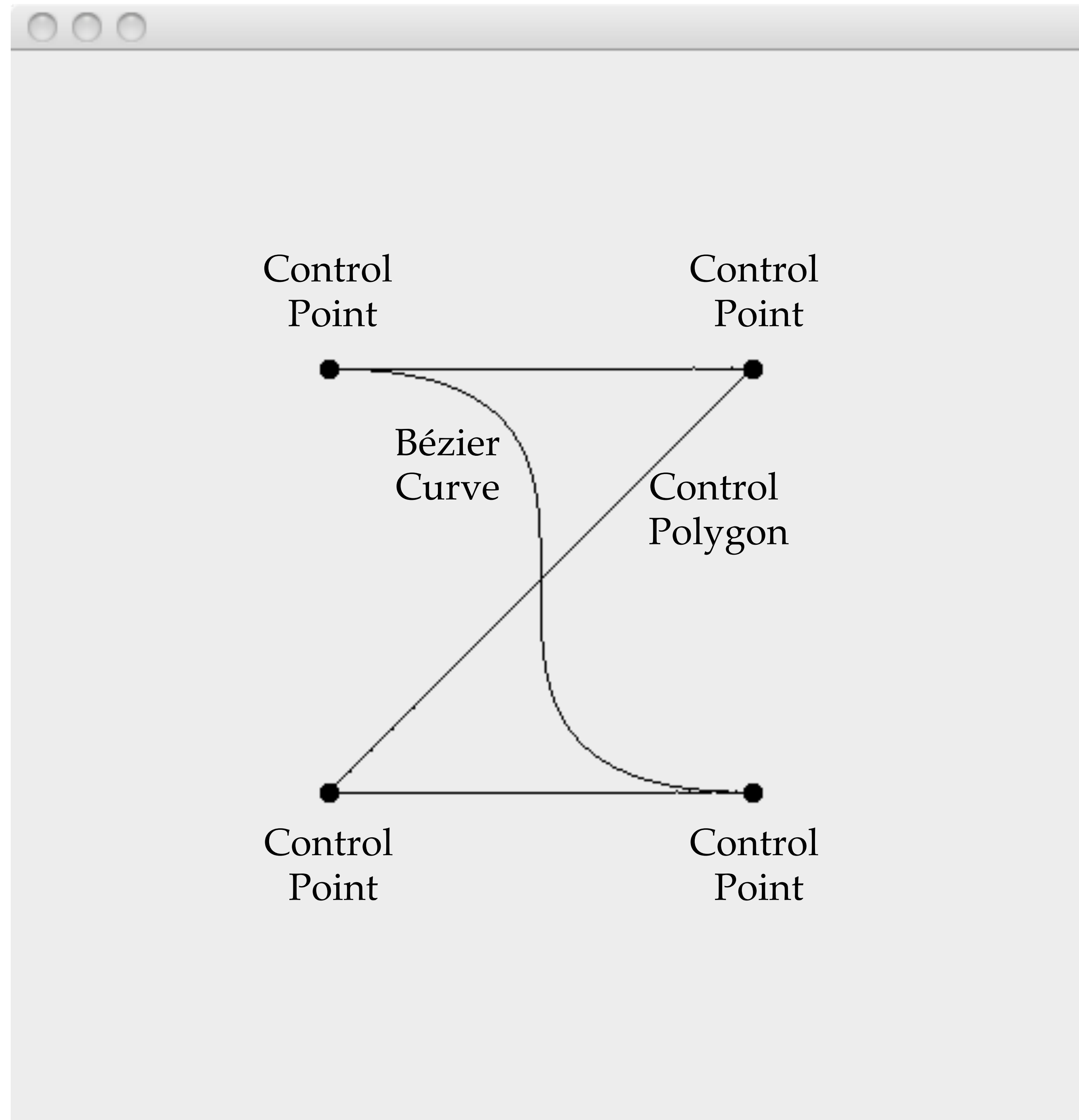
```
int N_PTS = 3;
Point bezPoints[N_PTS][N_PTS];

void DrawBezier()
  { // DrawBezier()
  for (float t = 0.0; t <= 1.0; t += 0.01)
    { // parameter loop
    for (int diag = N_PTS-2; diag >= 0; diag--)
      { // diagonal loop
      for (int i = 0; i <= diag; i++)
         { // i loop
         int j = diag - i;
         bezPoints[i][j] = (1.0-t)*bezPoints[i][j+1] + t*bezPoints[i+1][j];
         } // i loop
      } // diagonal loop
      // set the pixel for this parameter value
      SetPixel(bezPoints[0][0];
    } // parameter loop
  } // DrawBezier()
```

de Casteljau Example



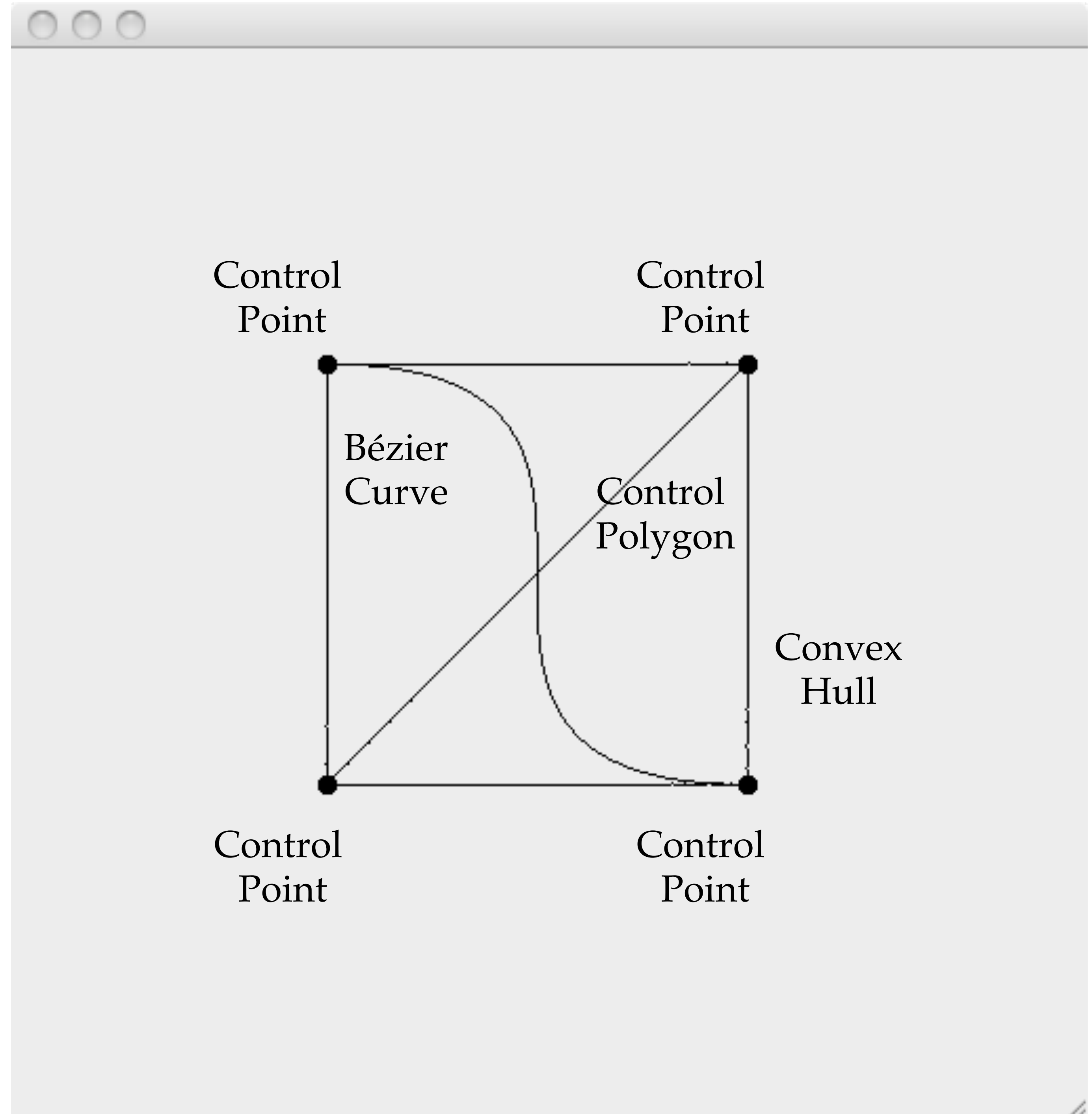
Bézier Control Polygons

- The points are the control points
- The lines are the control polygon
  - although they're actually a polyline



Bézier Convex Hull

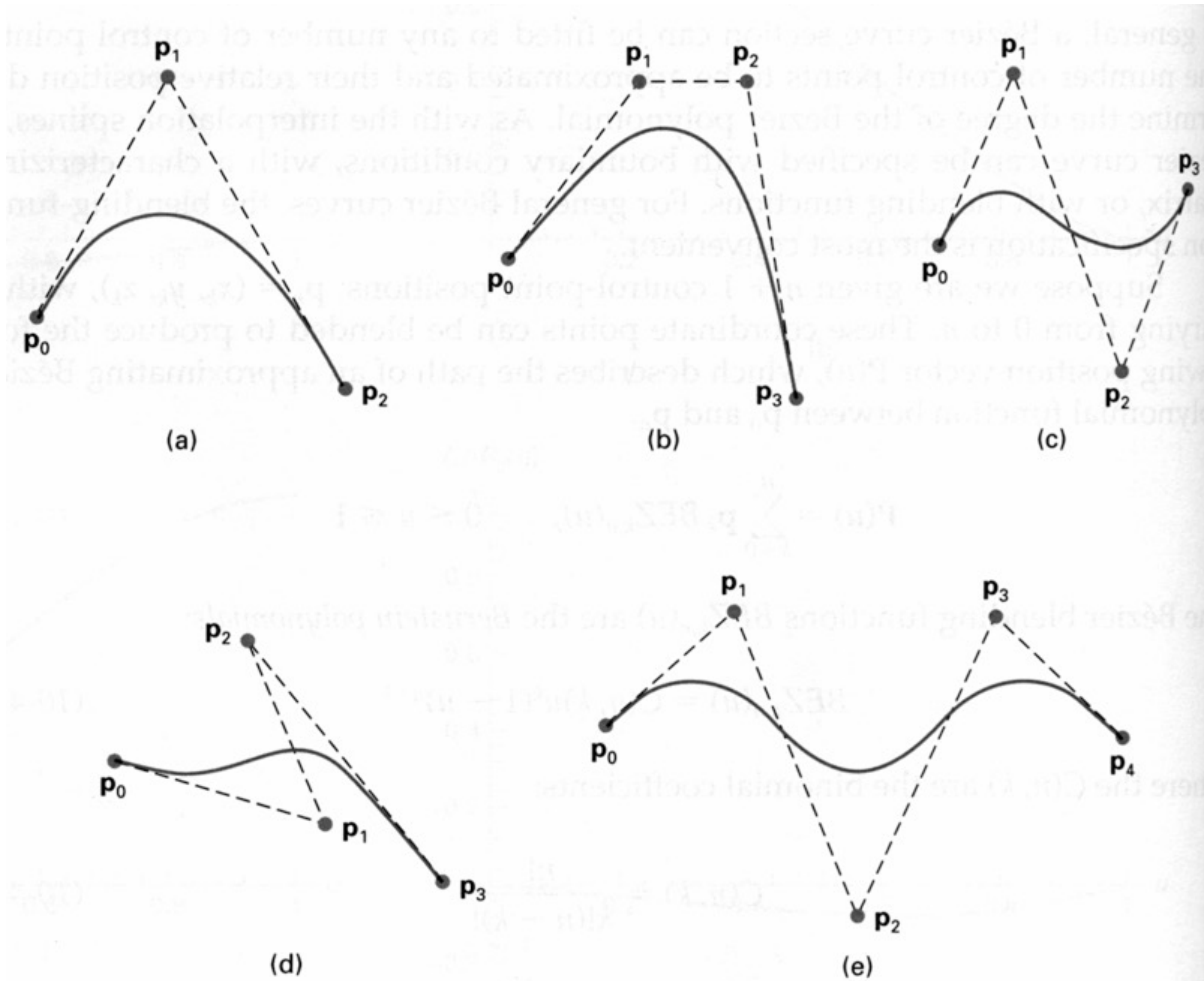
- Bézier uses linear interpolation
  - i.e. along lines
- Which means:
  - it's bounded
  - by the convex hull



Properties of Béziers

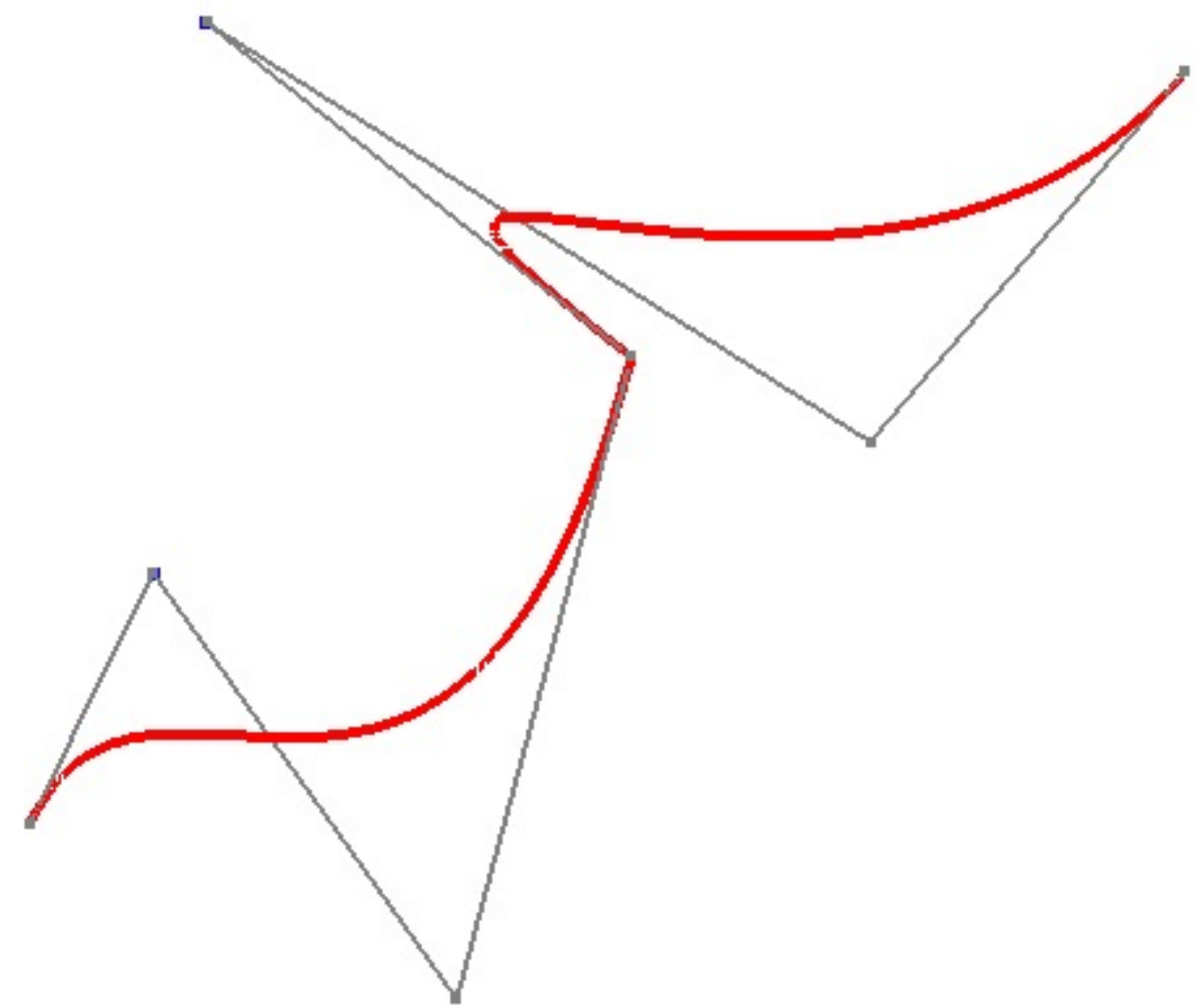
- Smooth - C^∞
- Interpolate through endpoints
- Bounded - convex hull property
- Efficient - repeated linear interpolation
- Numerically stable
- Easy to control

Some Examples

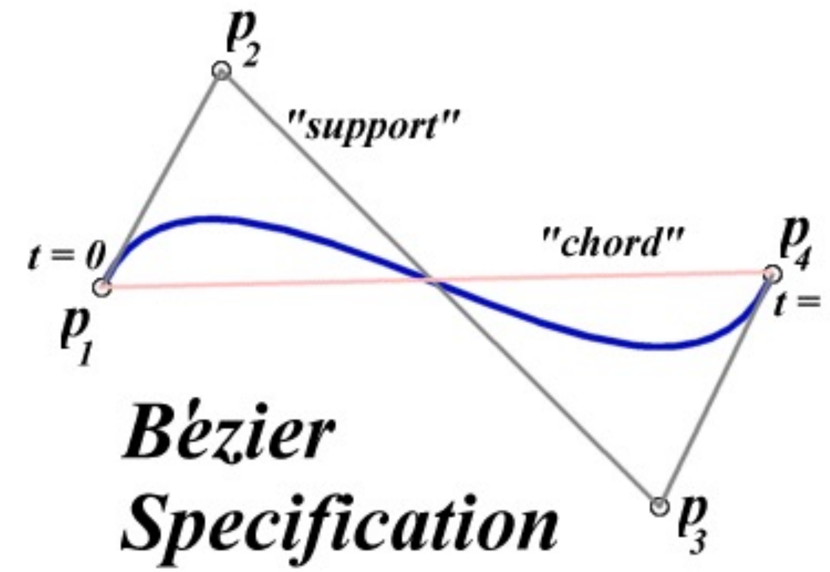


Piecewise Béziers

- Convenient, but
  - not C^1 continuous
  - we want slopes to match
  - unless we *want* sharp turns/corners



Cubic Bézier Curves



©T. Munzner, UBC

- Curves defined by *4* points
- Curve passes through two points
  - contained in *convex hull* of points

$$p_{00}(t) = \begin{bmatrix} p_{03} & p_{12} & p_{21} & p_{30} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t^1 \\ 1 \end{bmatrix}$$

Conversion

- Hermites can be converted to Béziers

$$\begin{bmatrix} p_{03} & p_{12} & p_{21} & p_{30} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t^1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_0 & \vec{x}_1' & \vec{x}_0' \end{bmatrix} \begin{bmatrix} -2 & 3 & 0 & 0 \\ 2 & -3 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t^1 \\ 1 \end{bmatrix}$$

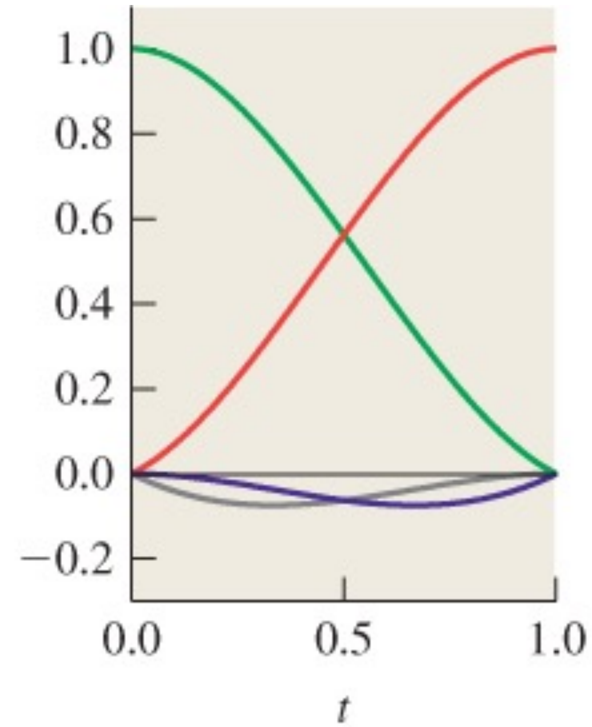
$$\begin{bmatrix} p_{03} & p_{12} & p_{21} & p_{30} \end{bmatrix} = \begin{bmatrix} x_1 & x_0 & \vec{x}_1' & \vec{x}_0' \end{bmatrix} \begin{bmatrix} -2 & 3 & 0 & 0 \\ 2 & -3 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}^{-1}$$

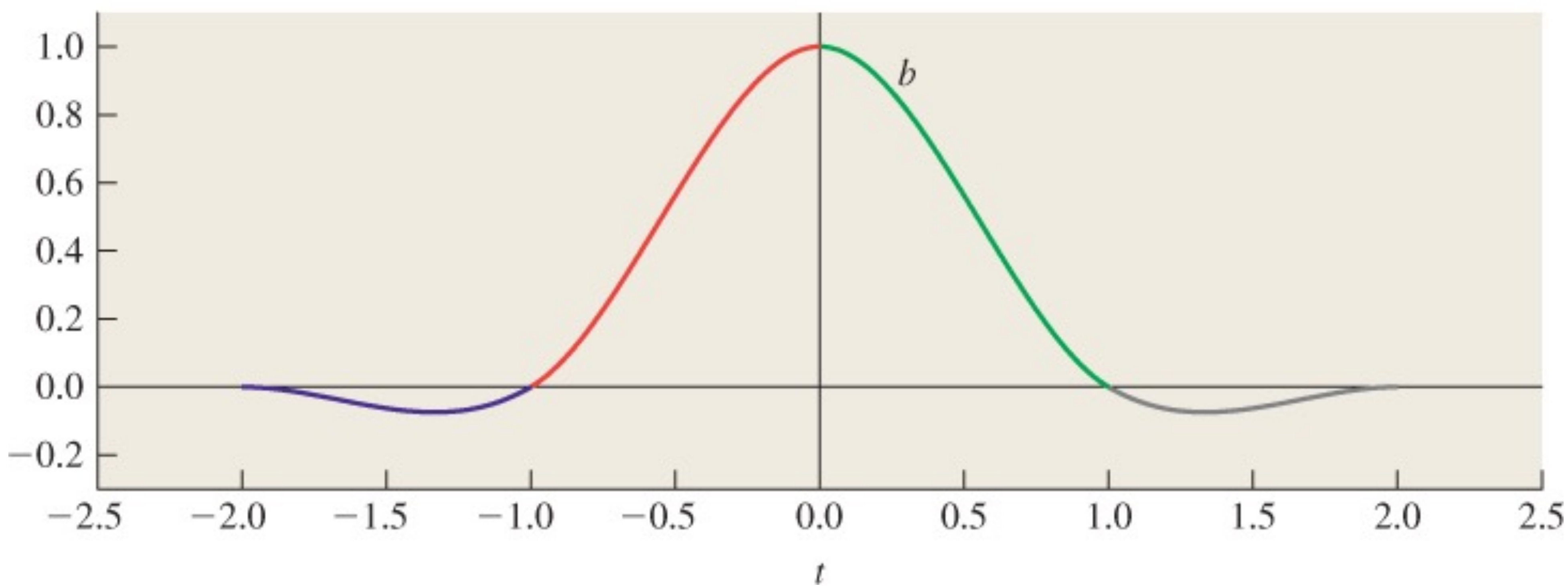
$$\begin{bmatrix} p_{03} & p_{12} & p_{21} & p_{30} \end{bmatrix} = \begin{bmatrix} x_1 & x_0 & \vec{x}_1' & \vec{x}_0' \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & -3 & 0 \end{bmatrix}$$

Catmull-Rom Splines

- Basically a fixed form of Hermites
- Interpolate through *every* point
- Vectors chosen are based on the points
  - $\vec{v}_i = \frac{1}{3}(p_{i+1} - p_{i-1})$
- Except at the endpoints
  - $\vec{v}_0 = \frac{2}{3}(p_1 - p_0)$
  - $\vec{v}_n = \frac{2}{3}(p_n - p_{n-1})$

Catmull-Rom Weights





Terminology

- Notice we have a *sequence* of points
- Which defines a sequence of curves
- And we reuse points along the sequence
- At integer parameter values called *knots*
- We can generalise this further
- With cubic B-splines

Cubic B-splines

- Generalise Bézier/Hermite/Catmull-Rom
- Reuse individual control points (knots)
- More efficient
- Identical to Bézier geometry matrix
  - except for last row

$$x(t) = \begin{bmatrix} x_{i-2} & x_{i-1} & x_i & x_{i+1} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} (t-i)^3 \\ (t-i)^2 \\ (t-i)^1 \\ 1 \end{bmatrix}$$

Non-Uniform B-Splines

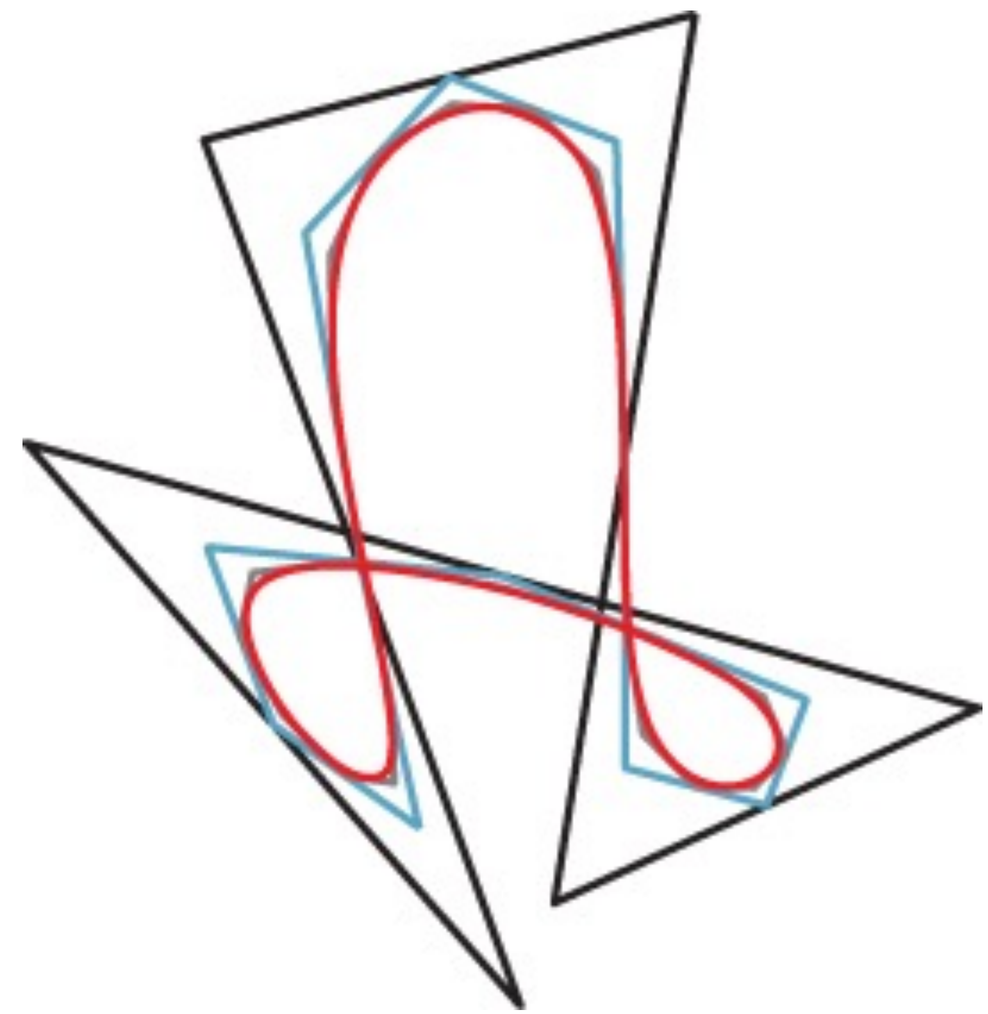
- Non-Uniform B-Splines
  - Knots are not evenly spaced
  - So we add t_i term to represent parameters
  - And we can repeat a point p_i
  - Which allows us to have sharp corners
- But we can't represent circles
  - Or other *conic* sections

NURBS

- Non-Uniform Rational B-Splines
- Add the homogeneous coordinate w
  - As a function of t
- Allows conic sections
- And survives projection!
- So we can project, then intersect

Subdivision Curves

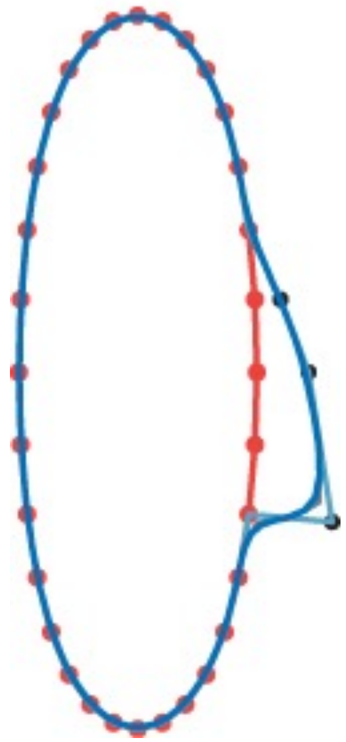
- Start with a polygon
- Shrink it inwards to get a smaller polygon
  - $e_i = (p_i + p_{i+1})$
  - $w_i = \frac{1}{2} p_i + \frac{1}{4} e_i + \frac{1}{4} e_{i+1}$
- In the limit, we get a curve



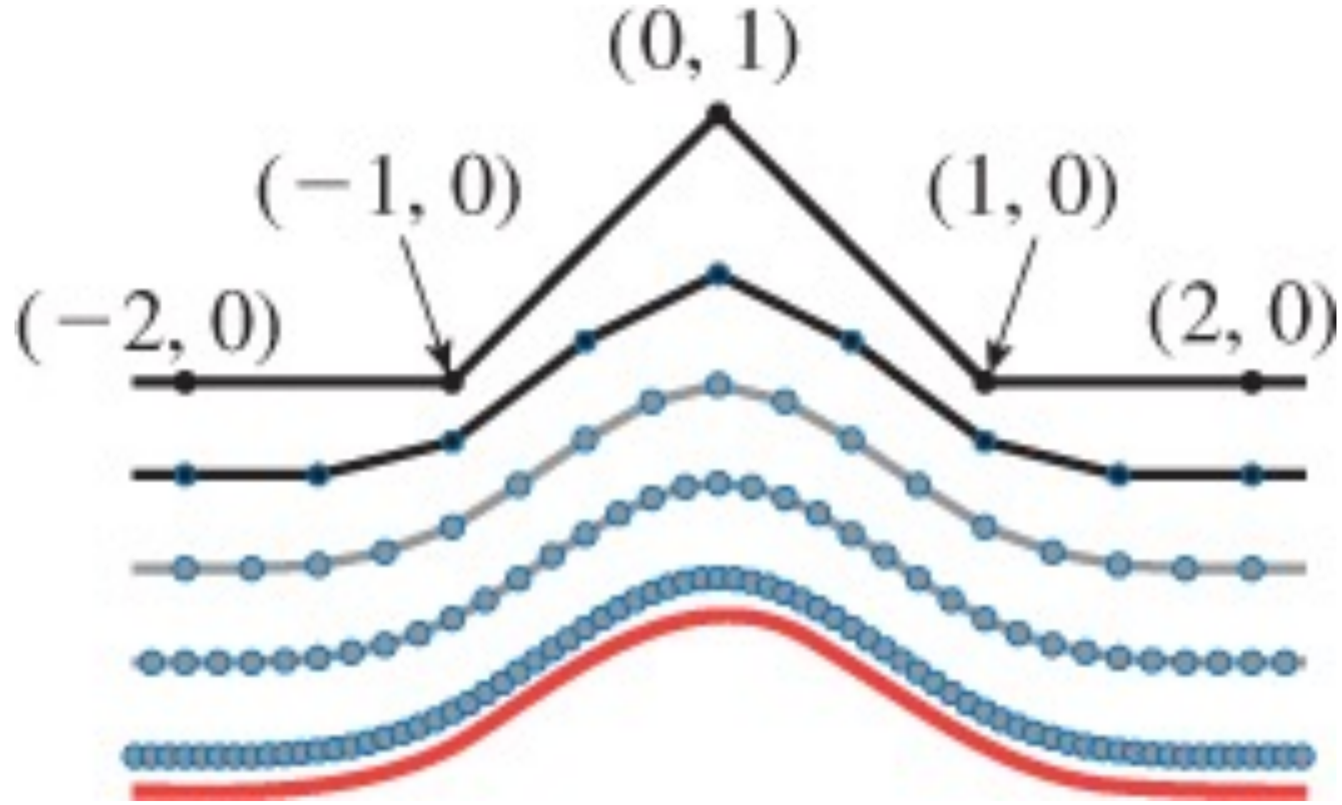
Editing Subdivision Curves

- Start with a simple shape
  - For example, an oval:

- At the second level
  - Move some nodes
  - To create a "nose"



Equivalence



- If we take the limit, we get a cubic curve
- And it's *exactly* the cubic B-spline