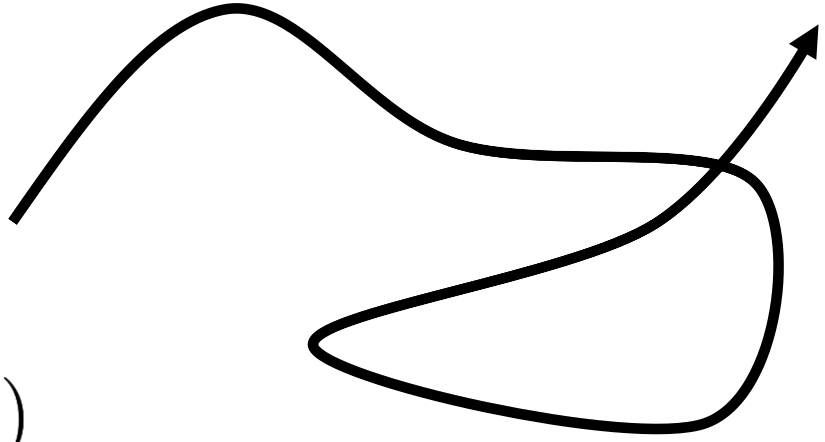
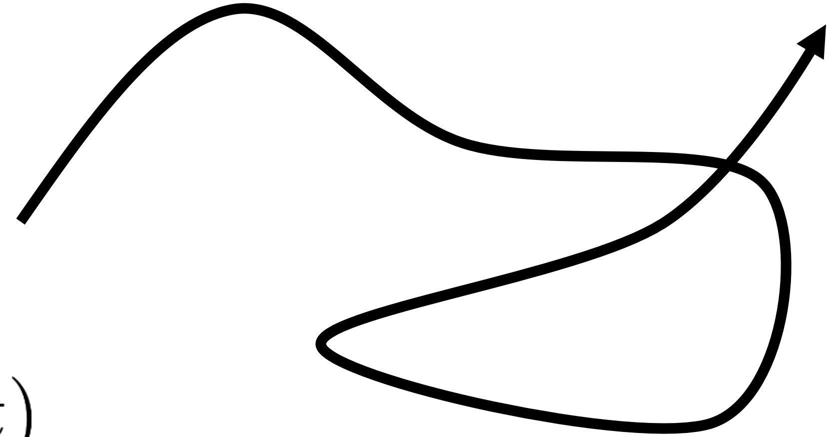
Curves and Splines

Curves in Spaces



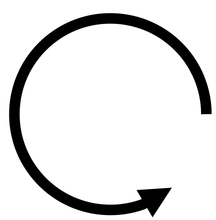
Parametric function $\gamma(t)$

Curves in Spaces



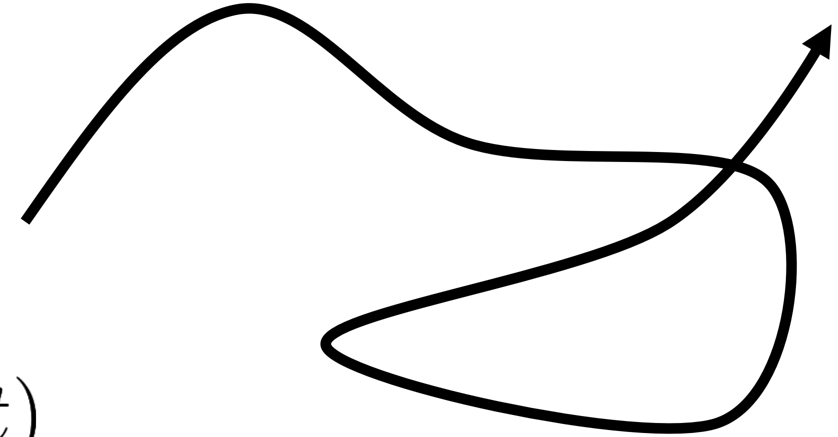
Parametric function $\gamma(t)$

Simple curves have well-known formulas:



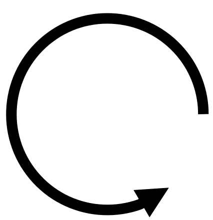
$$\gamma(t) = (\cos t, \sin t)$$

Curves in Spaces



Parametric function $\gamma(t)$

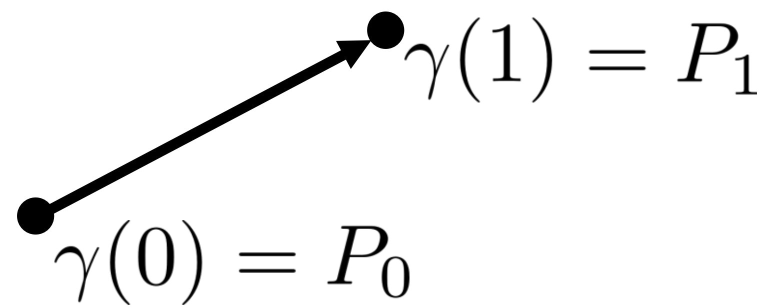
Simple curves have well-known formulas:

 $$\gamma(t) = (\cos t, \sin t)$$

What to do in general?

Linear Interpolation

Straight line segment between two points



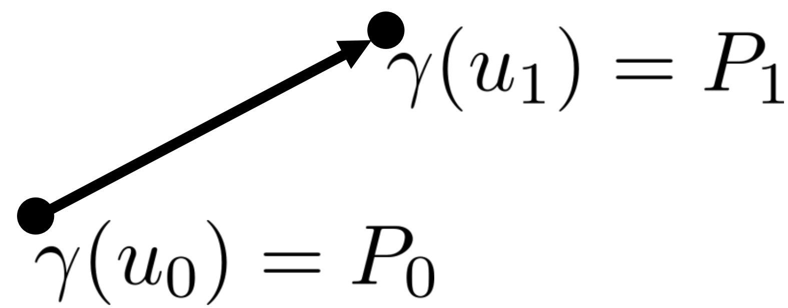
Linear Interpolation

Straight line segment between two points

$$\gamma(t) = P_0 + t(P_1 - P_0)$$
$$\gamma(t) = (1-t)P_0 + tP_1$$

Linear Interpolation

Straight line segment between two points

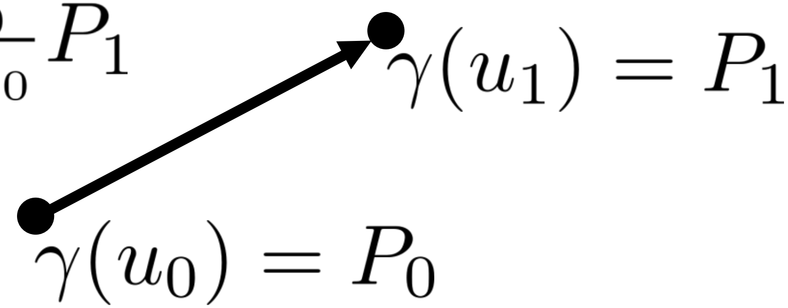


Also works for arbitrary parameterization

Linear Interpolation

Straight line segment between two points

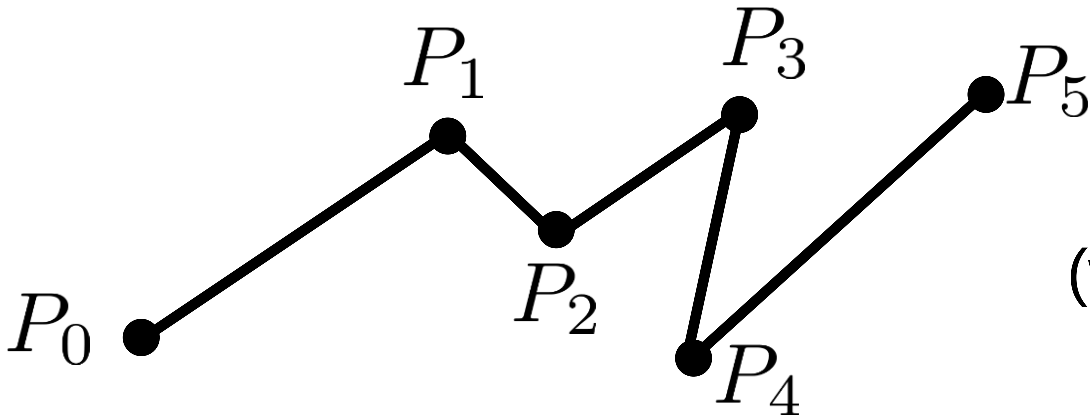
$$\gamma(t) = \frac{u_1 - t}{u_1 - u_0} P_0 + \frac{t - u_0}{u_1 - u_0} P_1$$



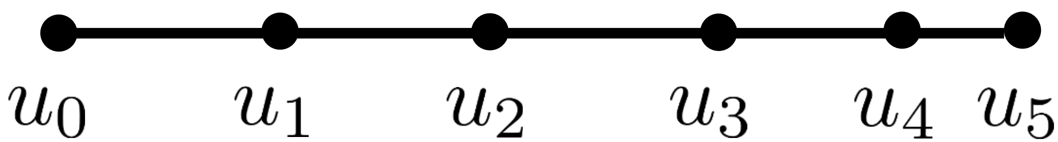
Also works for arbitrary parameterization

Piecewise Linear Interpolation

Straight line segment between point list



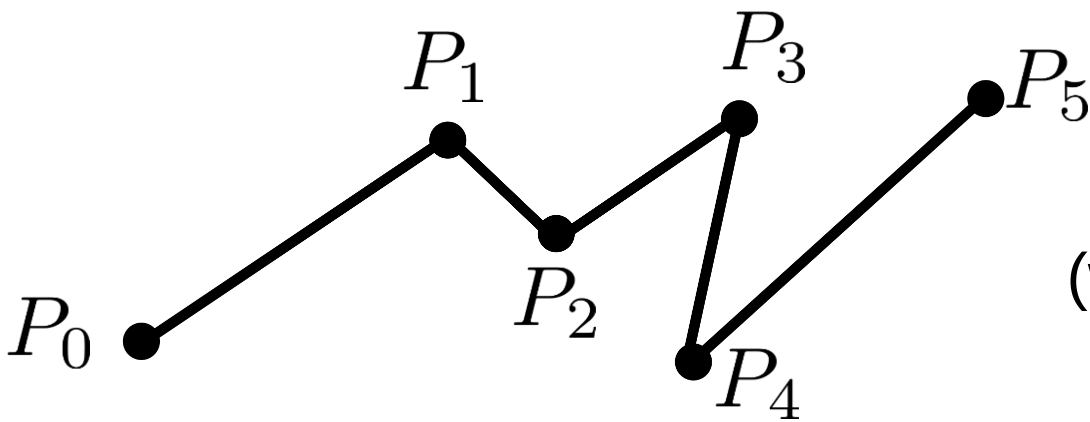
points in space
(where curves goes)



points in parameter space
(**knots**)
(how fast it goes)

Piecewise Linear Interpolation

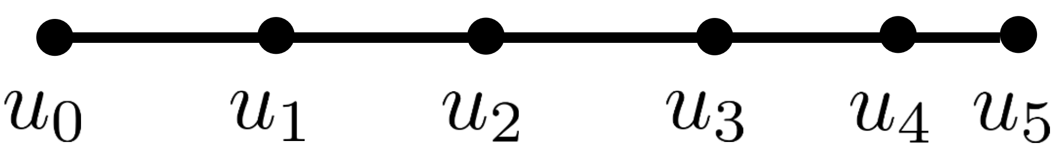
Straight line segment between point list



points in space
(where curves goes)

$$\gamma(t) = \frac{u_{i+1} - t}{u_{i+1} - u_i} P_i + \frac{t - u_i}{u_{i+1} - u_i} P_{i+1}$$

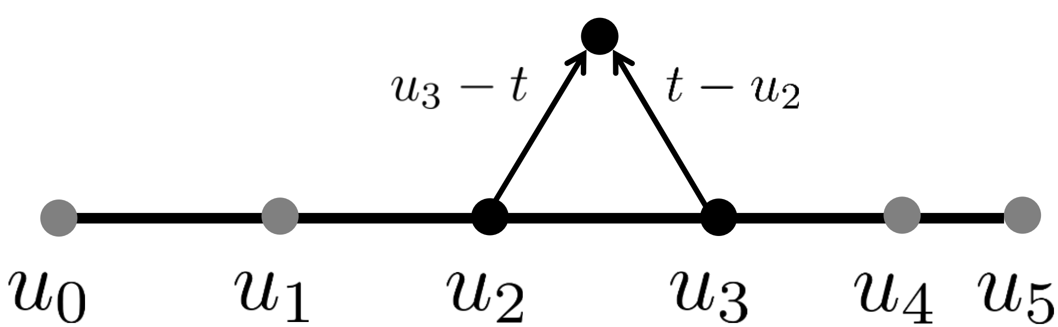
points in parameter space
(**knots**)
(how fast it goes)



Piecewise Linear Interpolation

"Pyramid Notation"

$$\gamma(t) = \frac{u_{i+1} - t}{u_{i+1} - u_i} P_i + \frac{t - u_i}{u_{i+1} - u_i} P_{i+1}$$

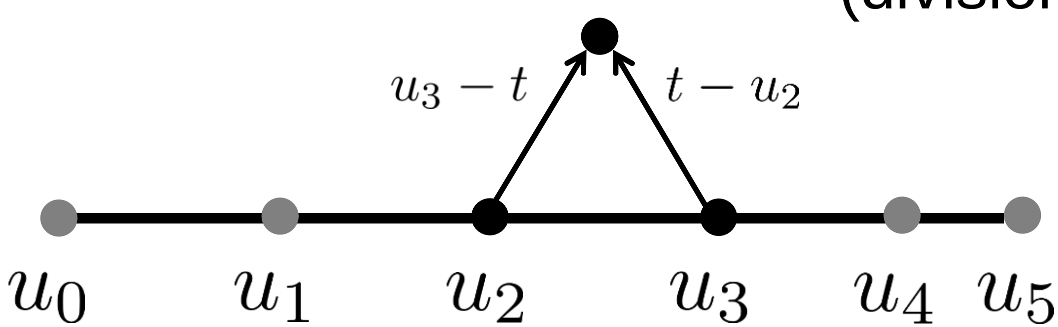


Piecewise Linear Interpolation

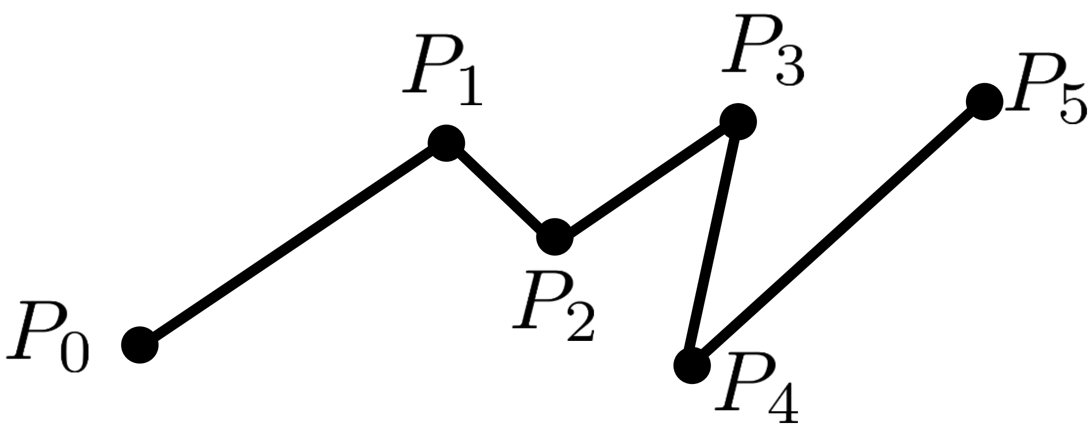
"Pyramid Notation"

$$\gamma(t) = \frac{u_{i+1}-t}{u_{i+1}-u_i} P_i + \frac{t-u_i}{u_{i+1}-u_i} P_{i+1}$$

(division by **sum** implicit)

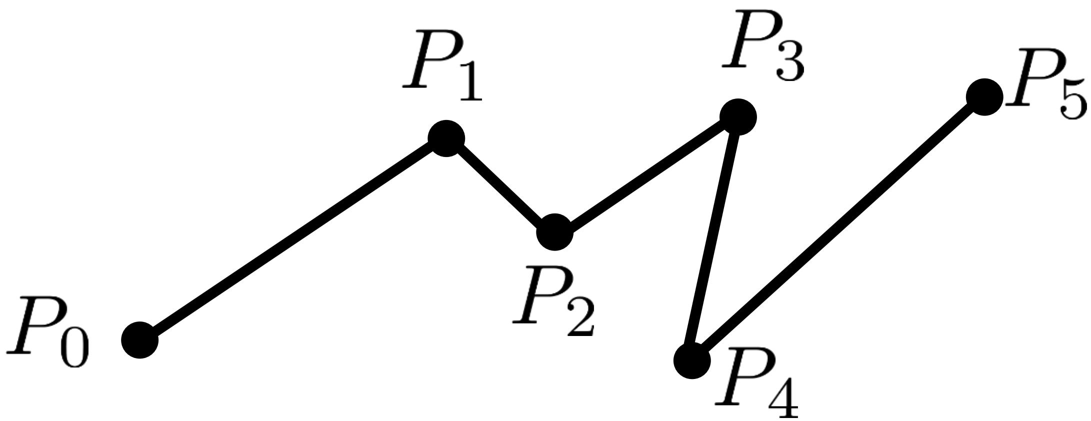


Piecewise Linear Interpolation



Easy, but “chunky” – only C^0

Piecewise Linear Interpolation



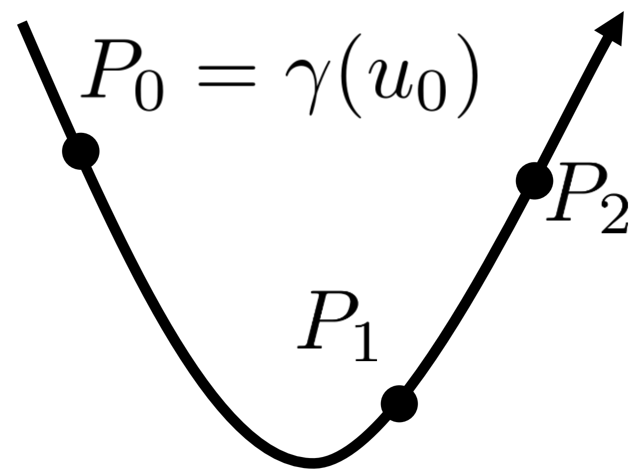
Easy, but “chunky” – only C^0

Continuity notation: C^n means continuous after taking n derivatives

Lagrange Interpolation

Given some points, find polynomial

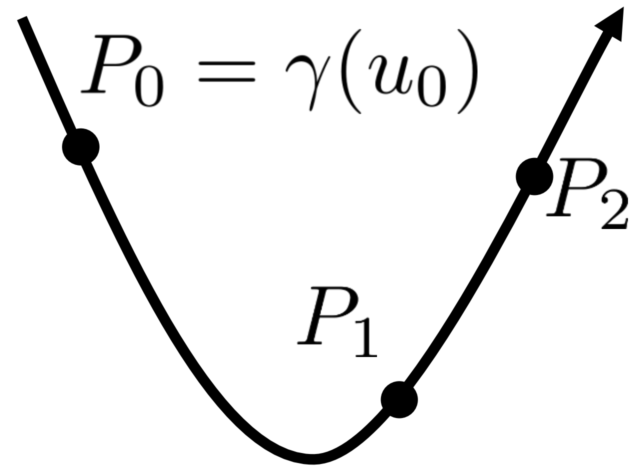
$$\gamma(t) = \left[\begin{array}{c} a_x + b_x t + c_x t^2 + \ldots \\ a_y + b_y t + c_y t^2 + \ldots \end{array} \right]$$



Lagrange Interpolation

Given some points, find polynomial

$$\gamma(t) = \begin{bmatrix} a_x + b_x t + c_x t^2 + \dots \\ a_y + b_y t + c_y t^2 + \dots \end{bmatrix}$$

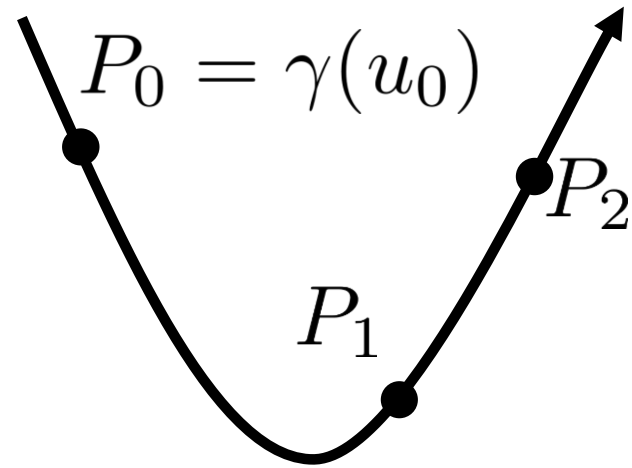


Notice: each coordinate is **linear combination** of a power of t

Lagrange Interpolation

Given some points, find polynomial

$$\gamma(t) = \begin{bmatrix} a_x & b_x & c_x & \cdots \\ a_y & b_y & c_y & \cdots \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \end{bmatrix}$$

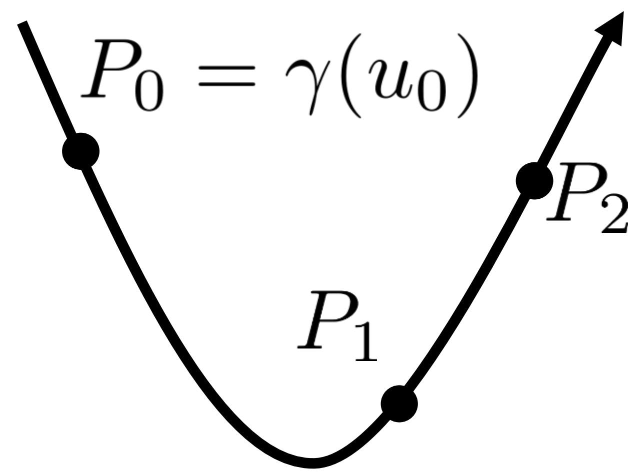


Notice: each coordinate is **linear combination** of a power of t

Lagrange Interpolation

Given some points, find polynomial

$$\gamma(t) = C_{2\times k} \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \end{bmatrix}_{k\times 1}$$

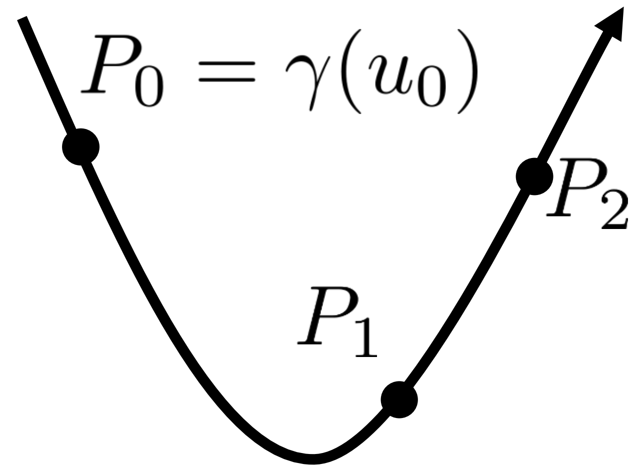


Notice: each coordinate is **linear combination** of a power of t

Lagrange Interpolation

Given some points, find polynomial

$$\gamma(t) = C_{2\times k} \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \end{bmatrix}_{k\times 1}$$

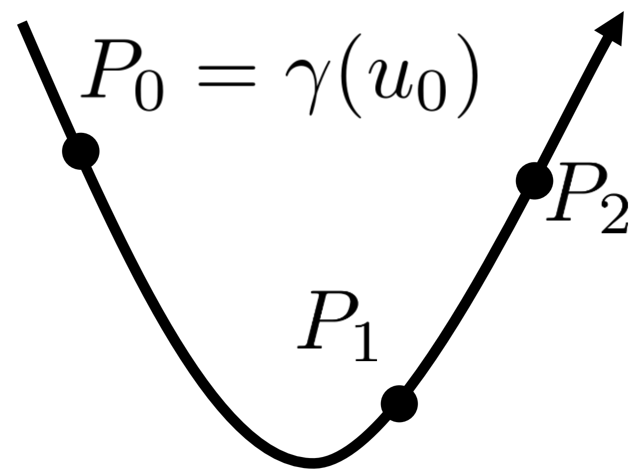


How to pick k?

Lagrange Interpolation

Given some points, find polynomial

$$P_i = C_{2\times k} \begin{bmatrix} 1 \\ u_i \\ u_i^2 \\ \vdots \end{bmatrix}_{k\times 1}$$

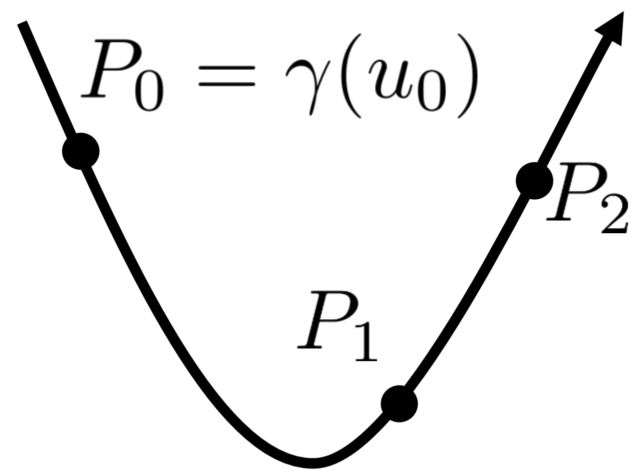


How to pick k? Use known points

Lagrange Interpolation

Given some points, find polynomial

$$\left[\begin{array}{c|c|c} P_0 & P_1 & P_2 \end{array}\right] = C_{2\times k} \begin{bmatrix} 1 & 1 & 1 \\ u_0 & u_1 & u_2 \\ u_0^2 & u_1^2 & u_2^2 \\ \vdots & \vdots & \vdots \end{bmatrix}_{k\times n}$$

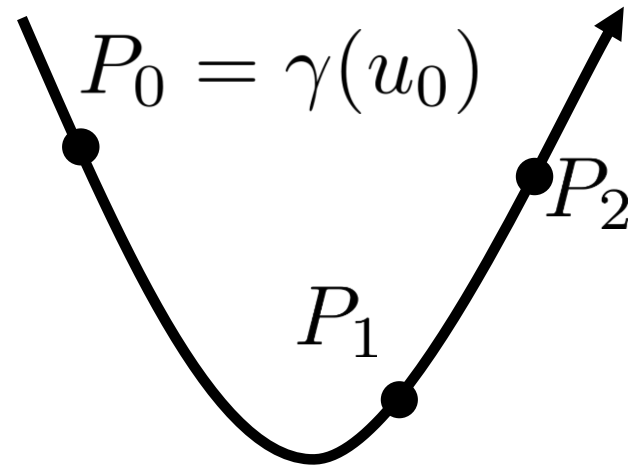


How to pick k? Use known points

Lagrange Interpolation

Given some points, find polynomial

$$\begin{bmatrix} P_0 & \Big| & P_1 & \Big| & P_2 \end{bmatrix} = C_{2\times k} \begin{bmatrix} 1 & 1 & 1 \\ u_0 & u_1 & u_2 \\ u_0^2 & u_1^2 & u_2^2 \\ \vdots & \vdots & \vdots \end{bmatrix}_{k\times n}$$

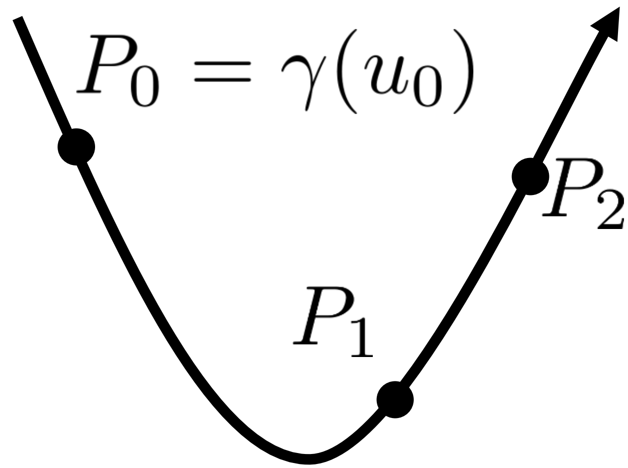


How to pick k? Use k = n

Lagrange Interpolation

Given some points, find polynomial

$$\left[\begin{array}{c|c|c} P_0 & P_1 & P_2 \end{array}\right] = C_{2\times n} \begin{bmatrix} 1 & 1 & 1 \\ u_0 & u_1 & u_2 \\ u_0^2 & u_1^2 & u_2^2 \end{bmatrix}_{n\times n}$$

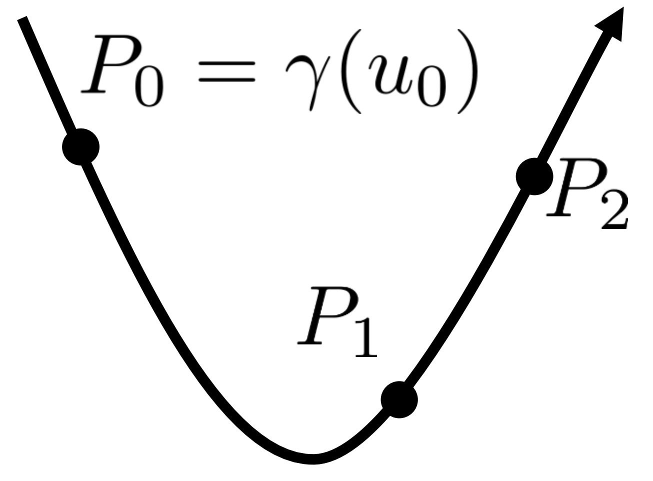


How to pick k? Use k = n

Lagrange Interpolation

Given some points, find polynomial

$$C_{2\times n} = \left[\begin{array}{c|c|c} P_0 & P_1 & P_2 \end{array}\right] \begin{bmatrix} 1 & 1 & 1 \\ u_0 & u_1 & u_2 \\ u_0^2 & u_1^2 & u_2^2 \end{bmatrix}^{-1}$$



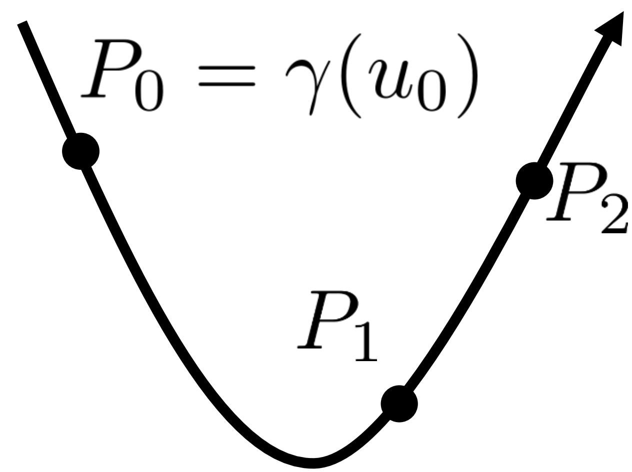
How to pick k? Use k = n

Lagrange Interpolation

Given some points, find polynomial

n points → degree (n-1)

- 2: linear interpolation
- 3: quadratic interp.

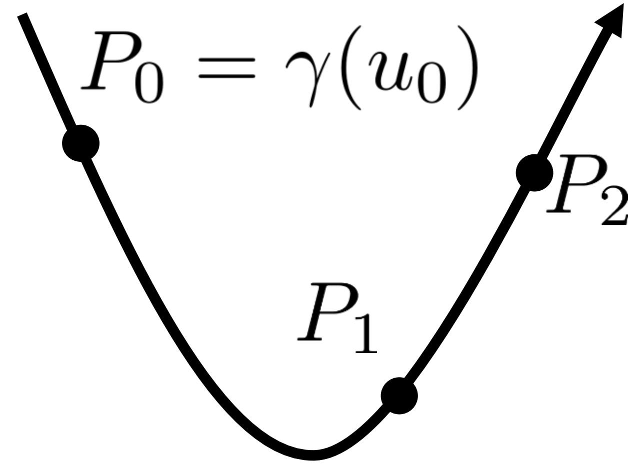


Lagrange Interpolation

Given some points, find polynomial

n points → degree (n-1)

- 2: linear interpolation
- 3: quadratic interp.



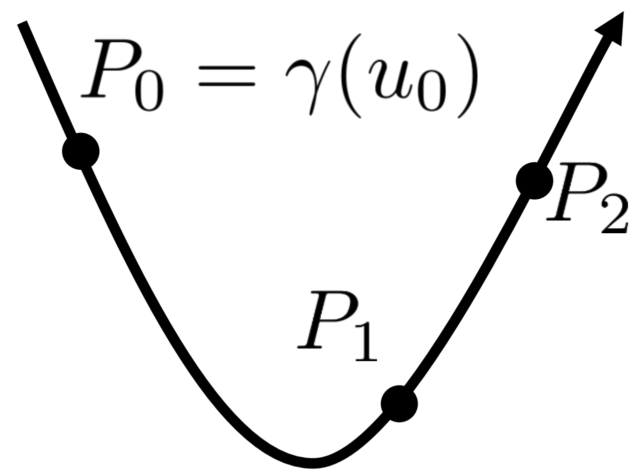
Curves are C^{n-2} smooth

Lagrange Interpolation

Given some points, find polynomial

n points → degree (n-1)

- 2: linear interpolation
- 3: quadratic interp.

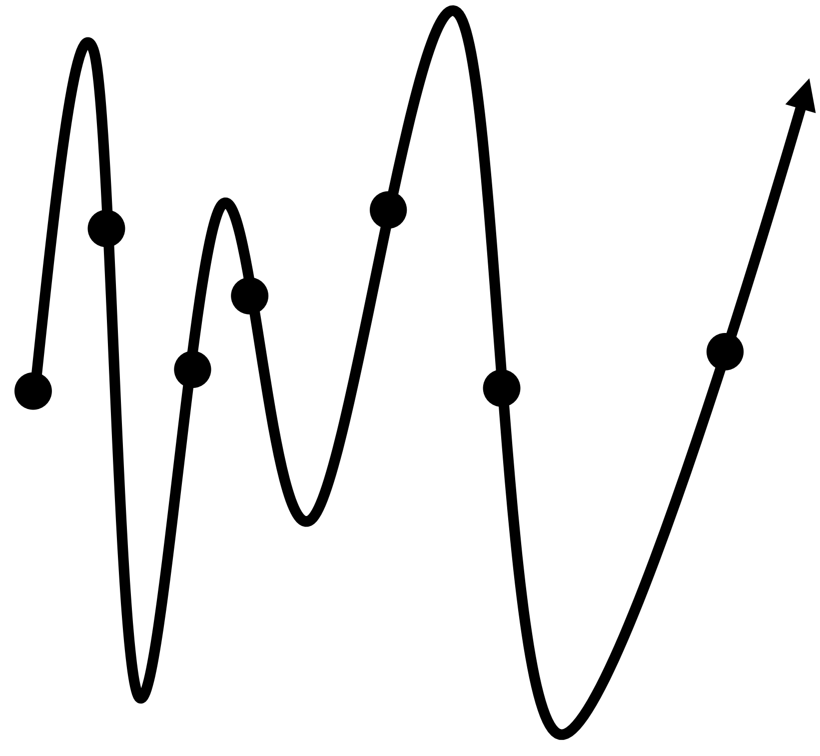


Curves are C^{n-2} smooth

What's the problem?

Lagrange Interpolation

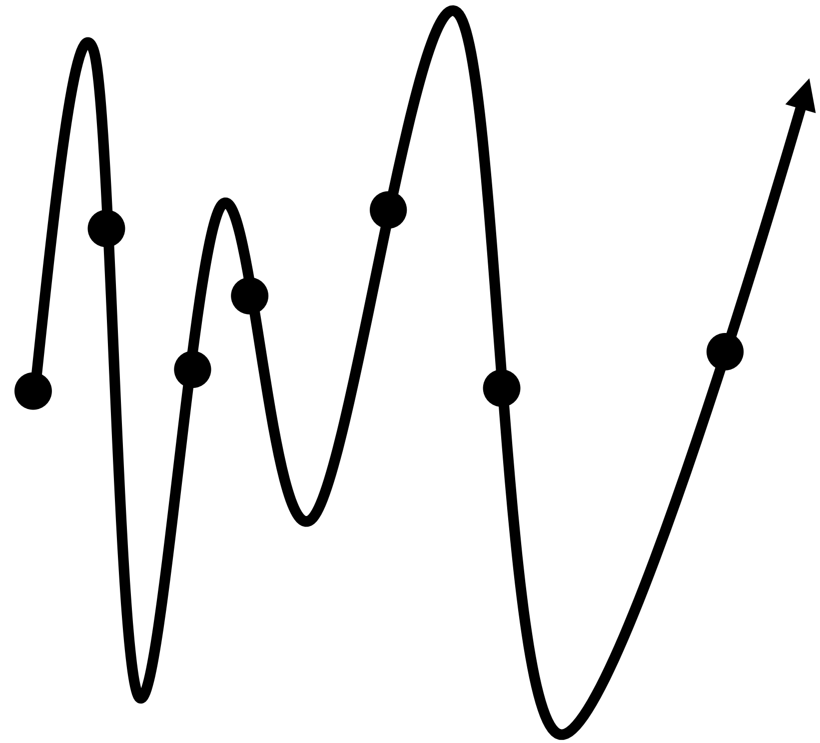
No oscillation control



Lagrange Interpolation

No oscillation control

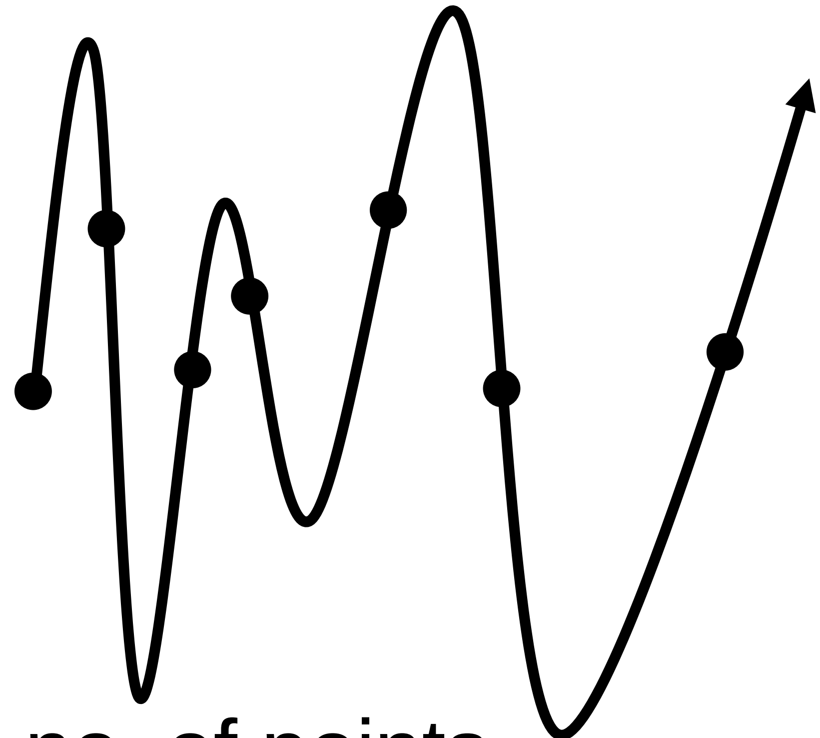
Worse as degree
becomes larger



Lagrange Interpolation

No oscillation control

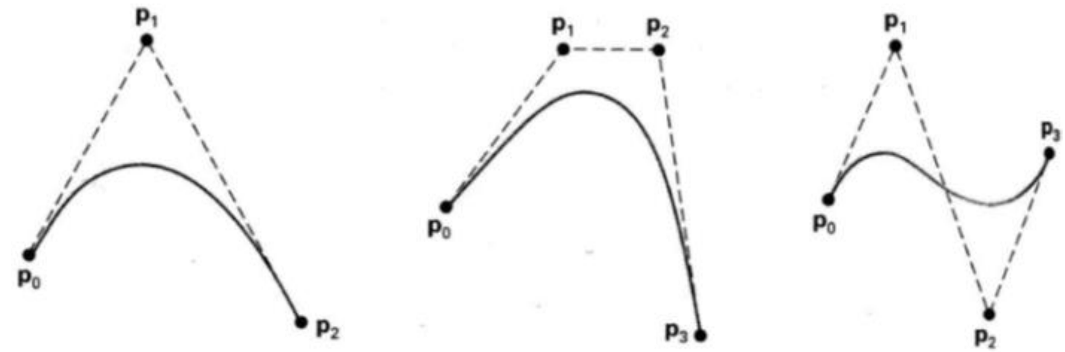
Worse as degree
becomes larger



Lagrange interpolation
not practical for large no. of points

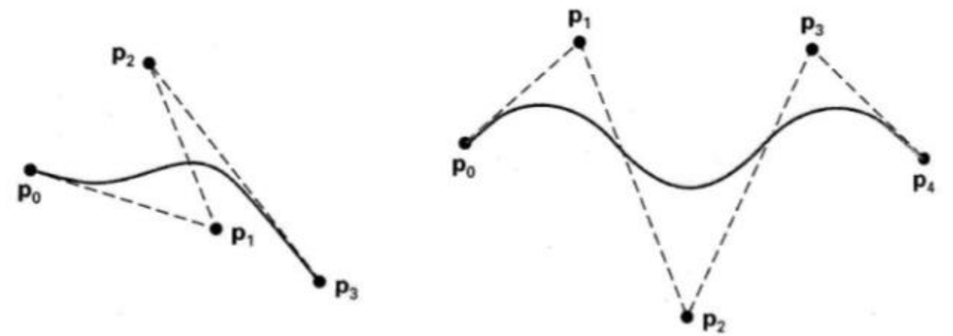
Introducing Splines

Bézier Curves

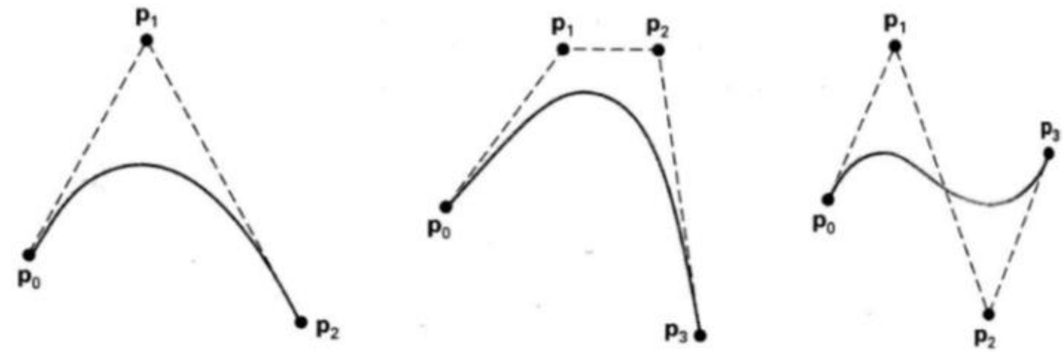


Spline building block

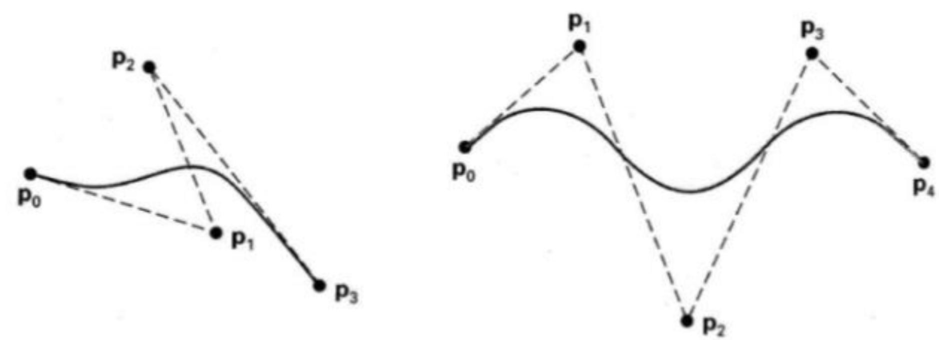
Polynomial



Bézier Curves



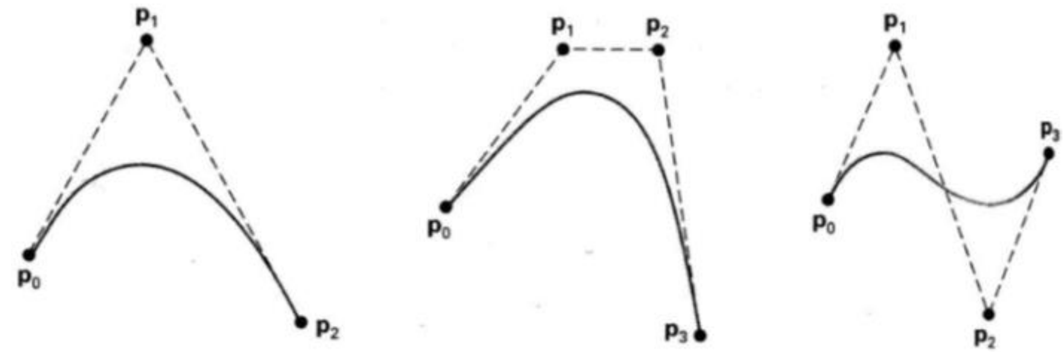
Spline building block



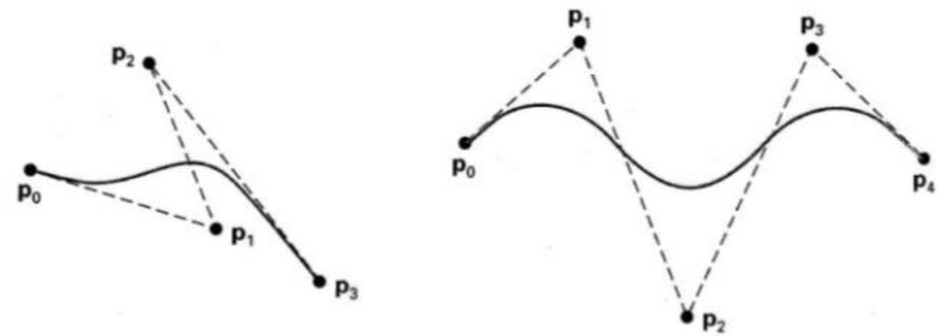
Polynomial

Variation-diminishing: curve lies in convex hull of points

Bézier Curves



Spline building block



Polynomial

Variation-diminishing: curve lies in convex hull of points

Cost: only interpolates endpoints

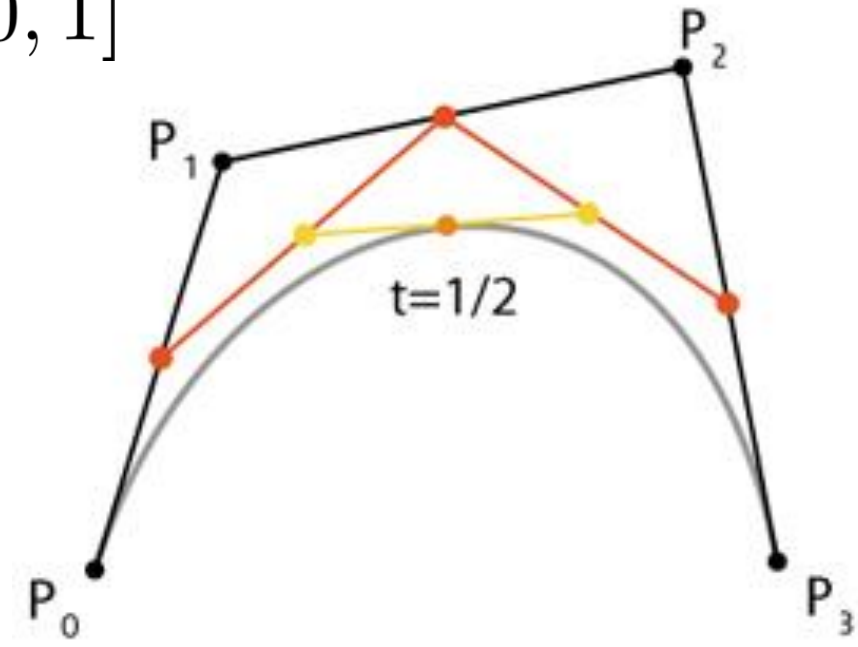
de Casteljau's Algorithm

Given:

- sequence of **control points** P_i
- **single** value of $t \in [0, 1]$

Computes:

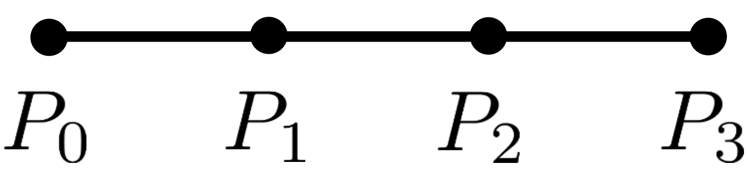
- location of $\gamma(t)$

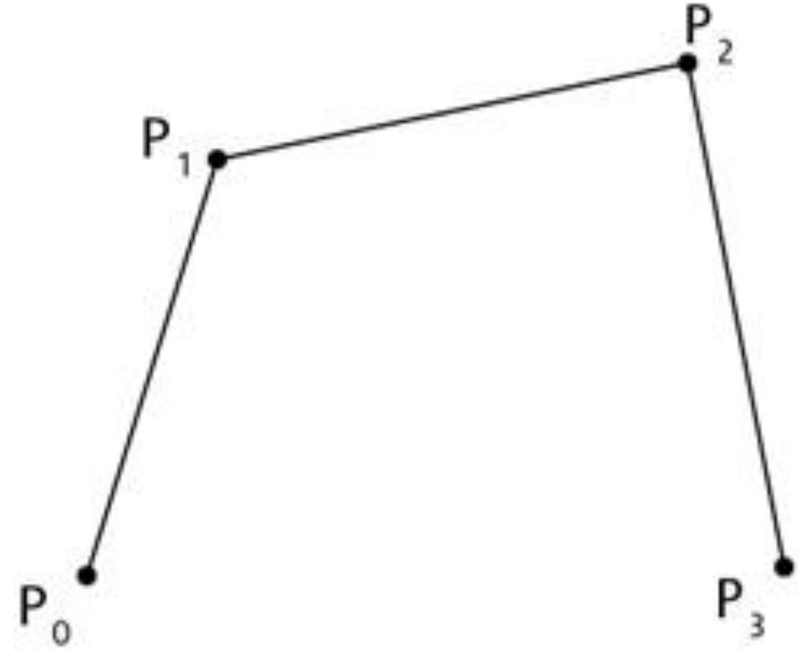


de Casteljau's Algorithm

Main idea: recursive linear interpolation

Start with four points – **control polygon**



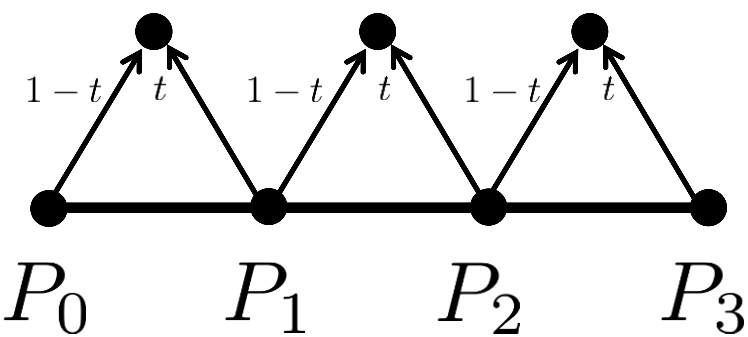


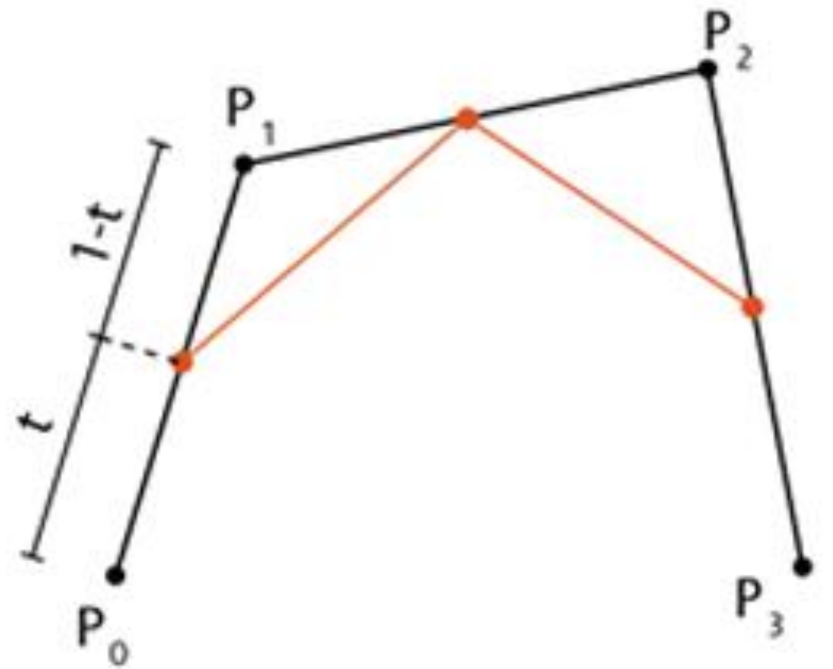
de Casteljau's Algorithm

Main idea: recursive linear interpolation

Start with four points – **control polygon**

Clip corners



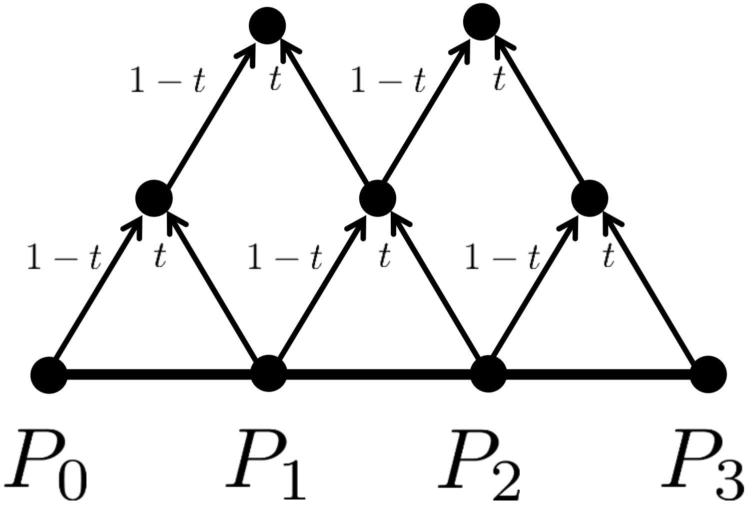


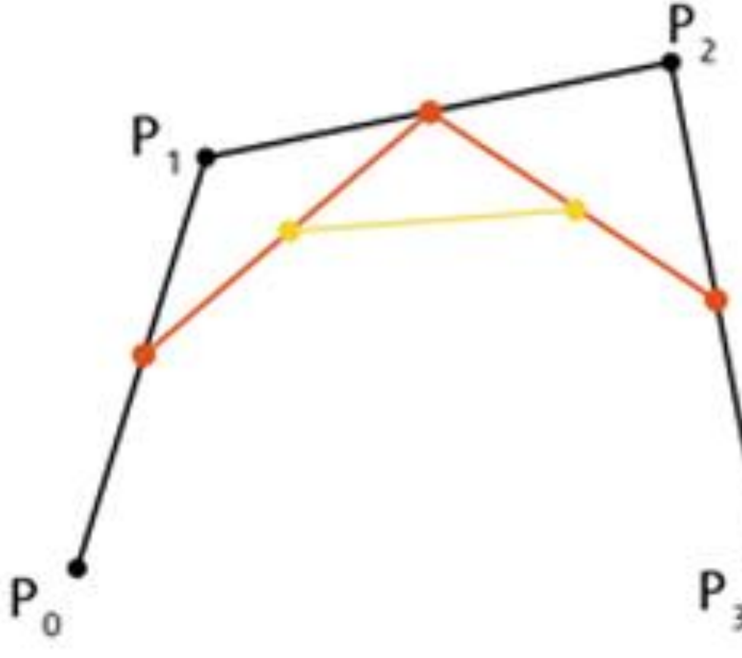
de Casteljau's Algorithm

Main idea: recursive linear interpolation

Start with four points – **control polygon**

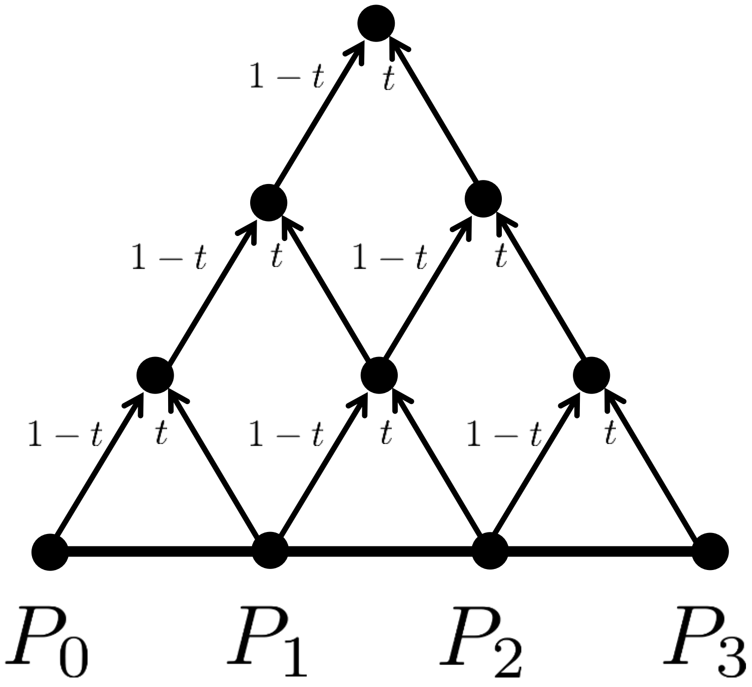
Clip corners

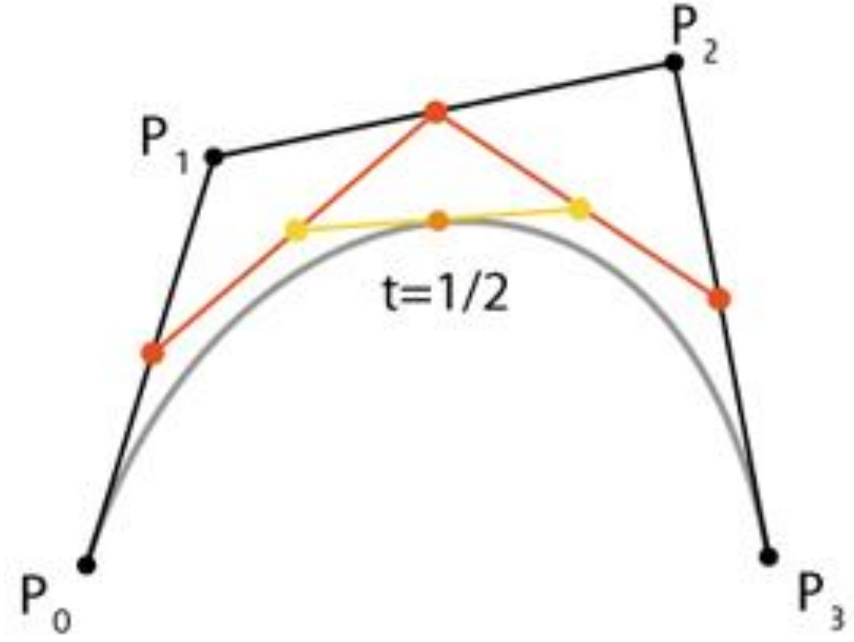




de Casteljau's Algorithm

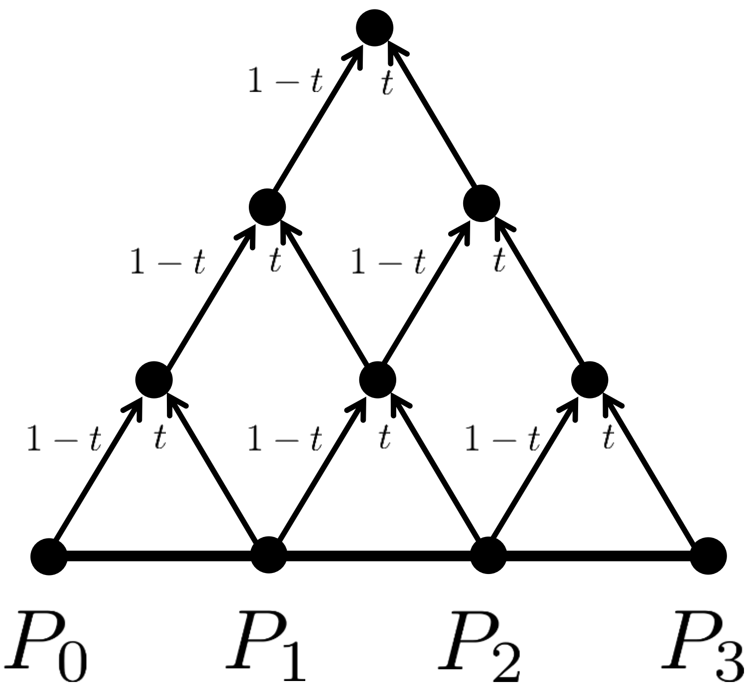
Four control points → cubic Bézier curve

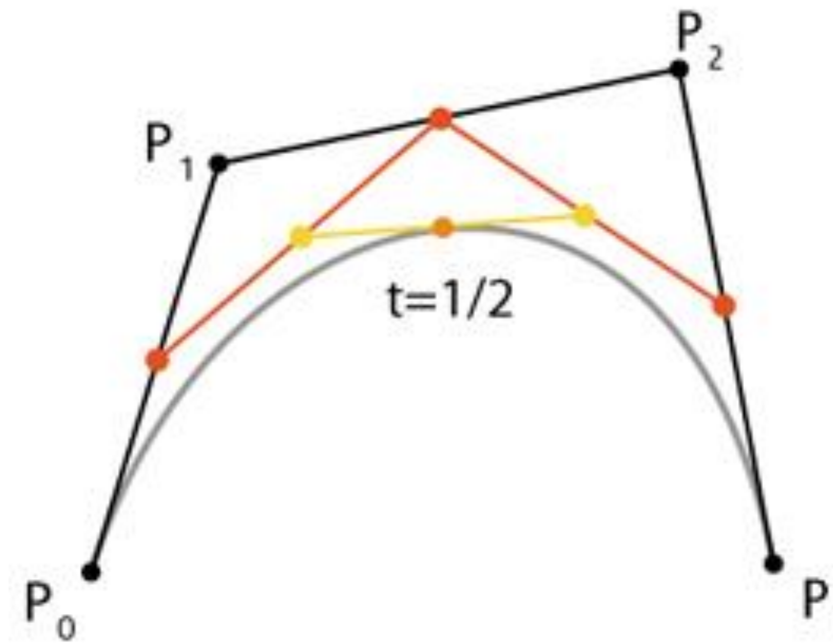




de Casteljau's Algorithm

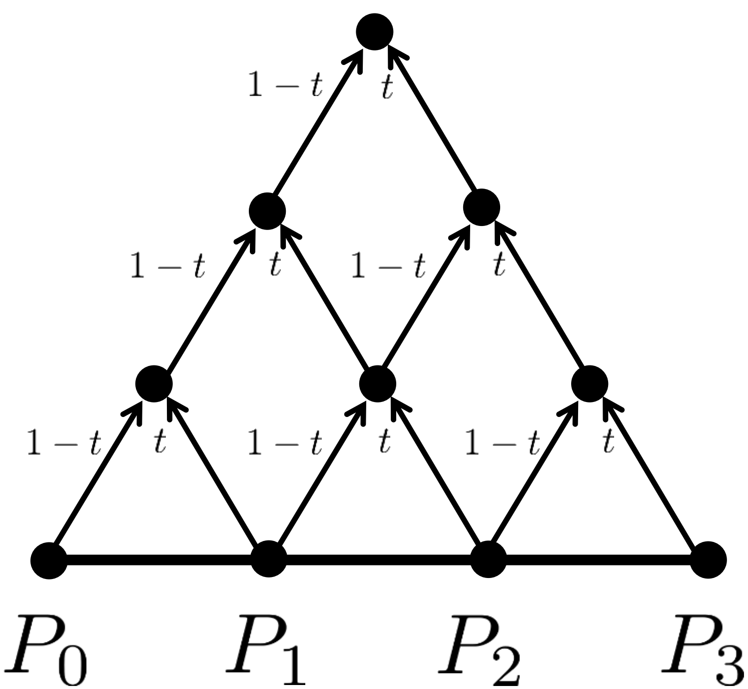
More control points → smoother curve
(more pyramid levels)





de Casteljau's Algorithm

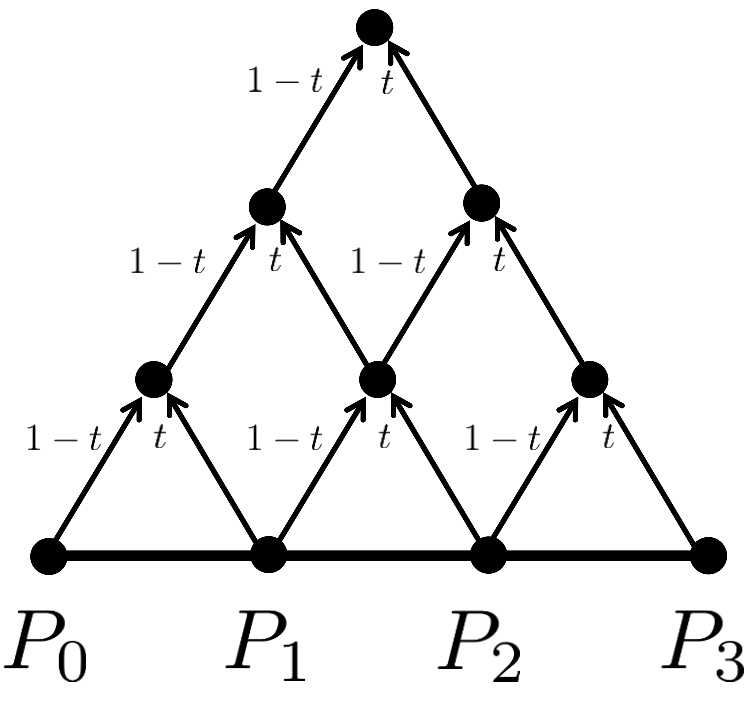
Time complexity?



de Casteljau's Algorithm

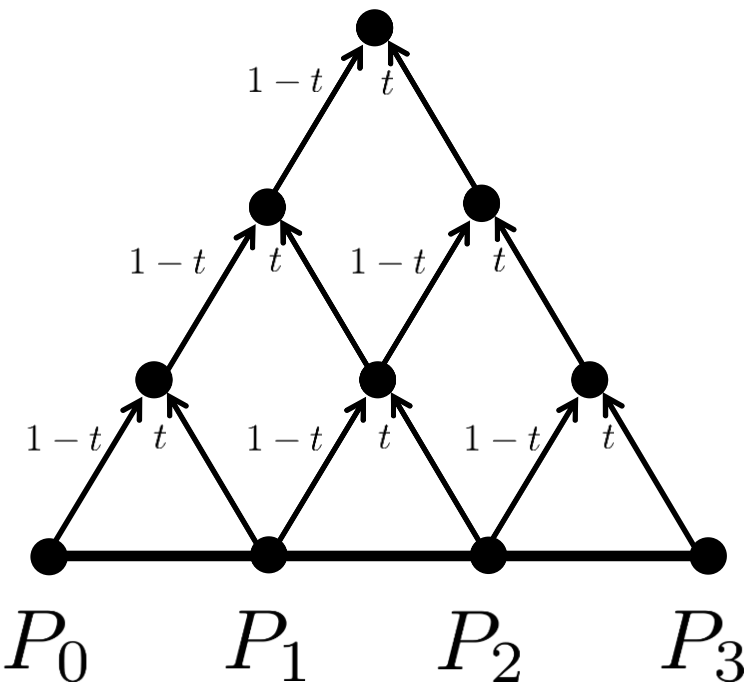
Time complexity?

O(n^2) for each evaluation



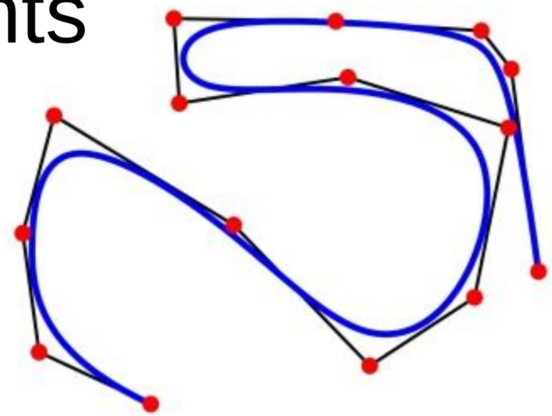
de Casteljau's Algorithm

Time complexity?



O(n^2) for each evaluation

Also, for long curve, may not want global influence of control points

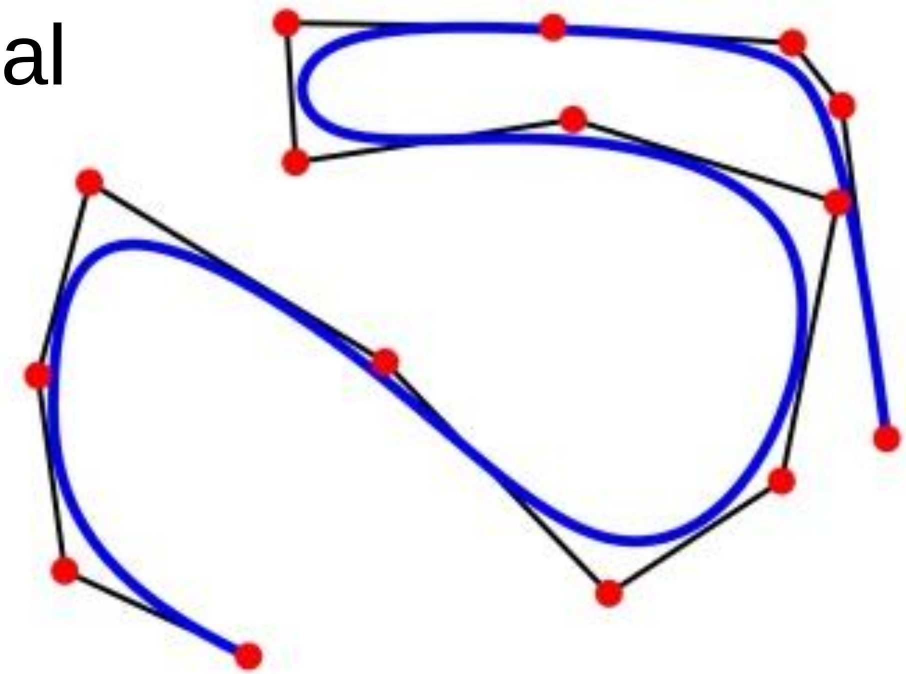


B-Splines ("Basis Splines")

Piecewise polynomial

- (cubic common)

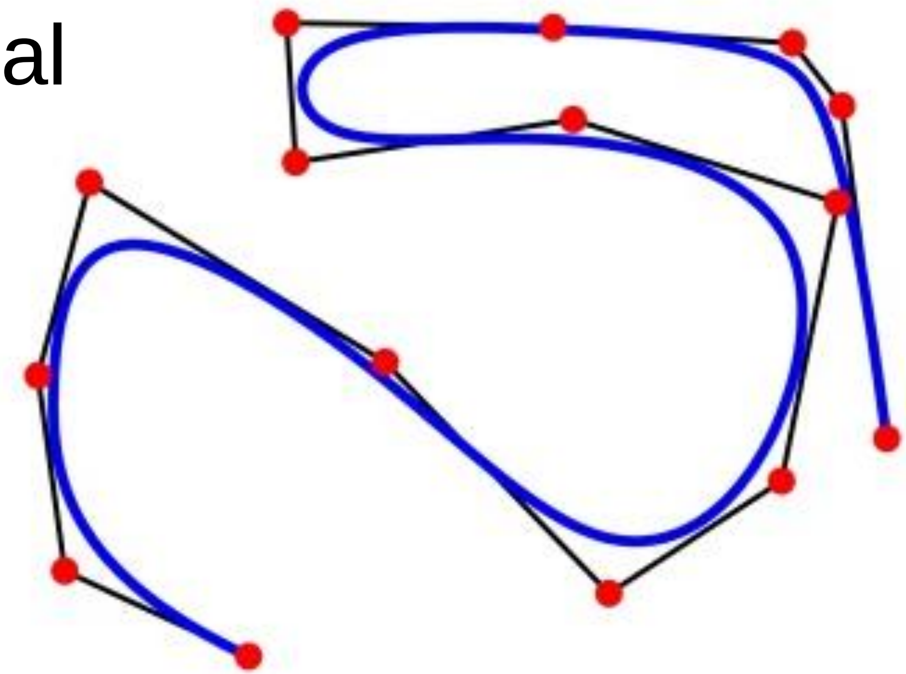
Used in Illustrator, Inkscape, etc



B-Splines ("Basis Splines")

Piecewise polynomial

- (cubic common)



Used in Illustrator,
Inkscape, etc

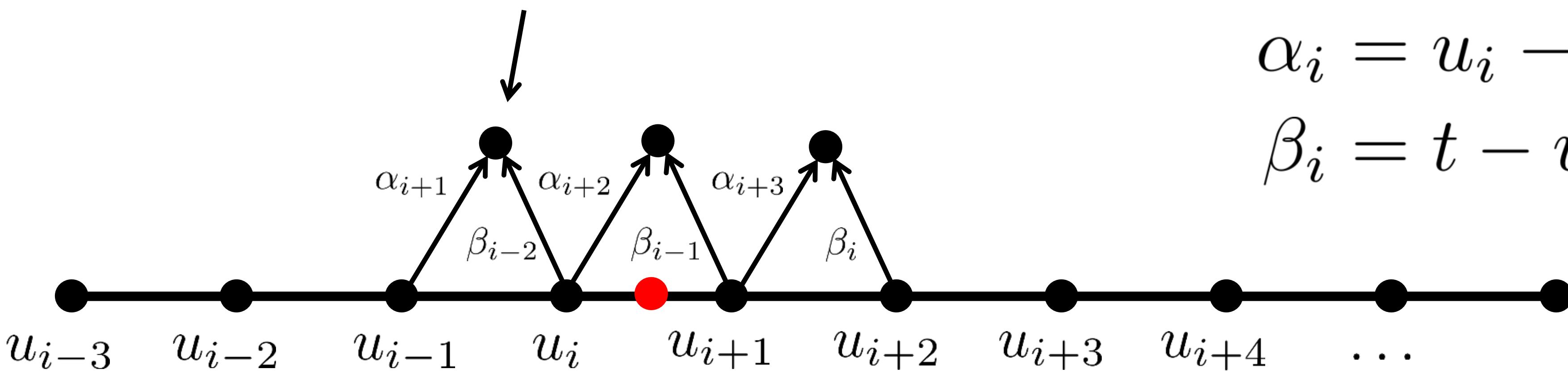
Arbitrary number of control points

- only first and last interpolated

de Boor’s Algorithm

Pyramid algorithm, like de Casteljau

$$\alpha_i = u_i - t$$
$$\beta_i = t - u_i$$



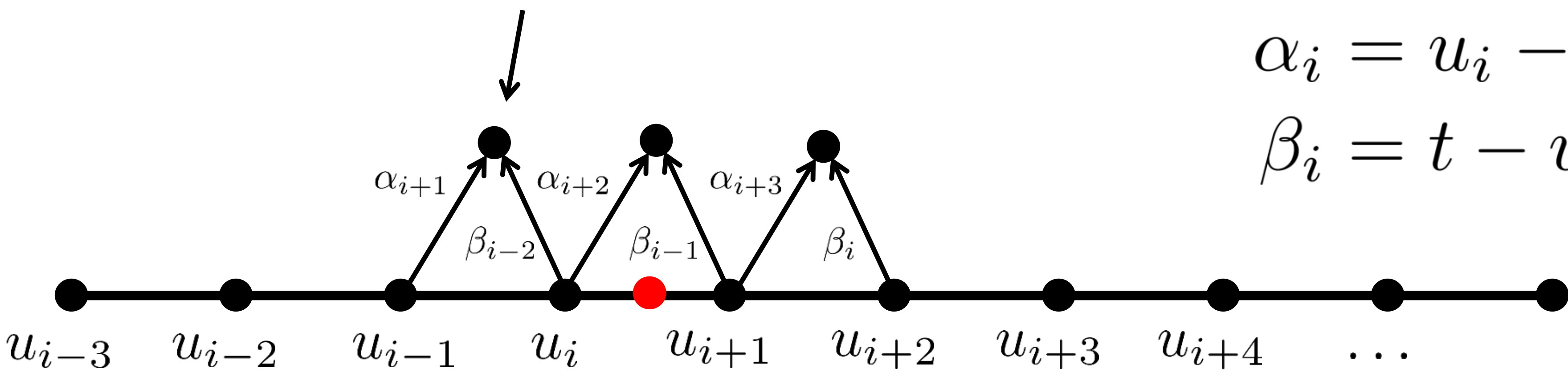
de Boor's Algorithm

Pyramid algorithm, like de Casteljau

$$\frac{\alpha_{i+1}}{\beta_{i-2}+\alpha_{i+1}} P_{i-1} + \frac{\beta_{i-2}}{\beta_{i-2}+\alpha_{i+1}} P_i$$

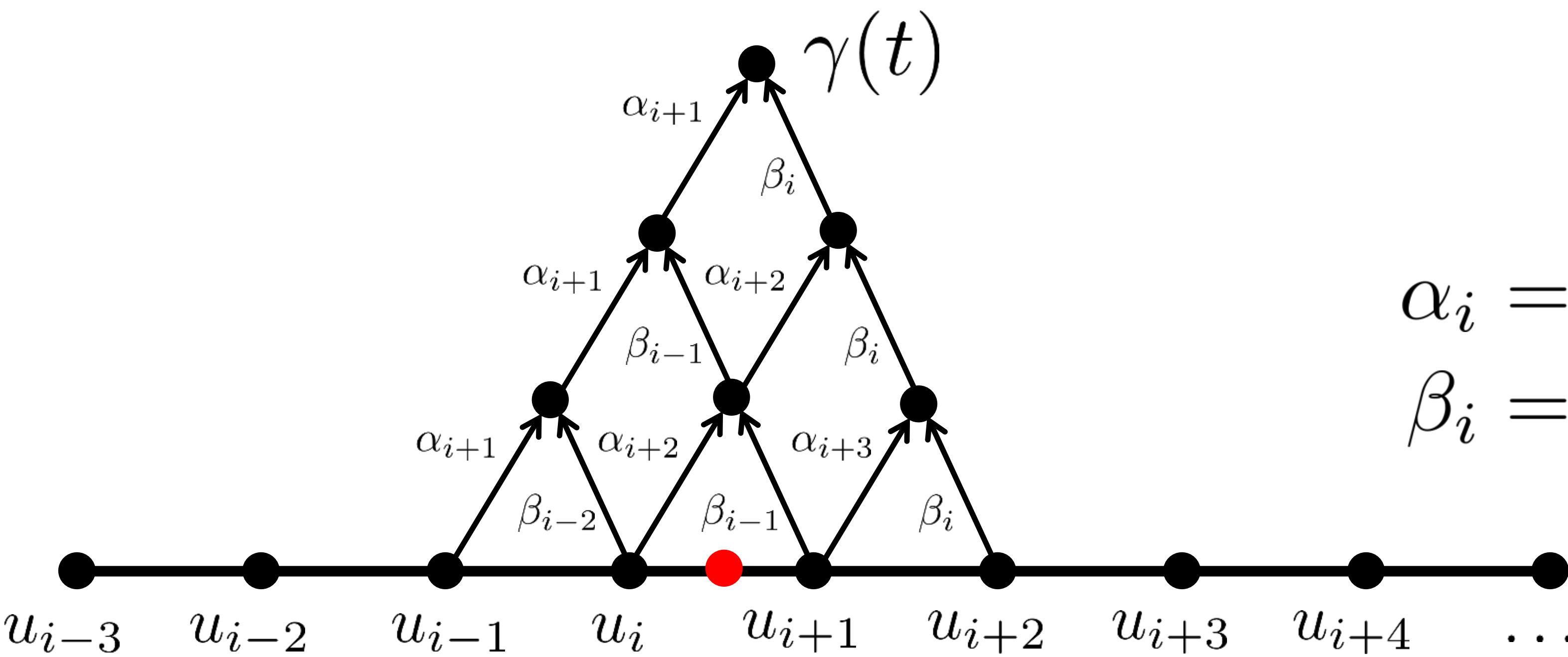
$$\alpha_i = u_i - t$$

$$\beta_i = t - u_i$$



de Boor's Algorithm

Pyramid algorithm, like de Casteljau

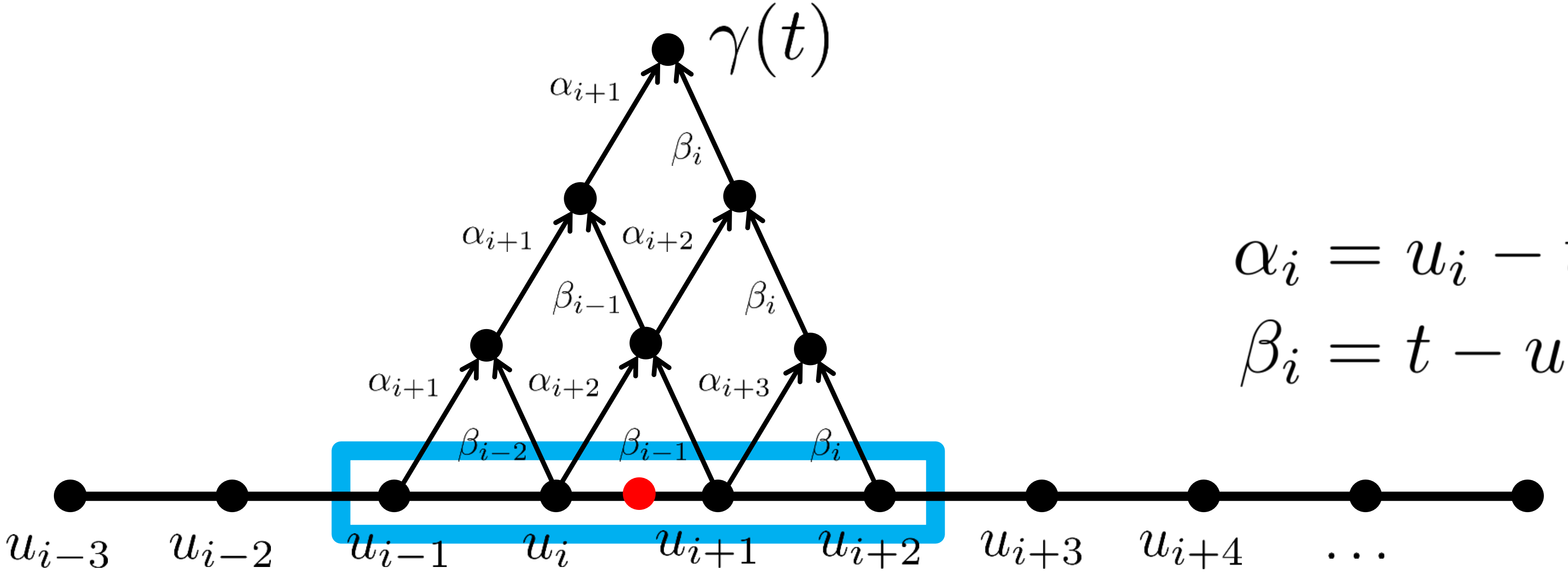


$$\alpha_i = u_i - t$$
$$\beta_i = t - u_i$$

de Boor's Algorithm

Pyramid algorithm, like de Casteljau

Final answer depends on **four** control pts



$$\alpha_i = u_i - t$$

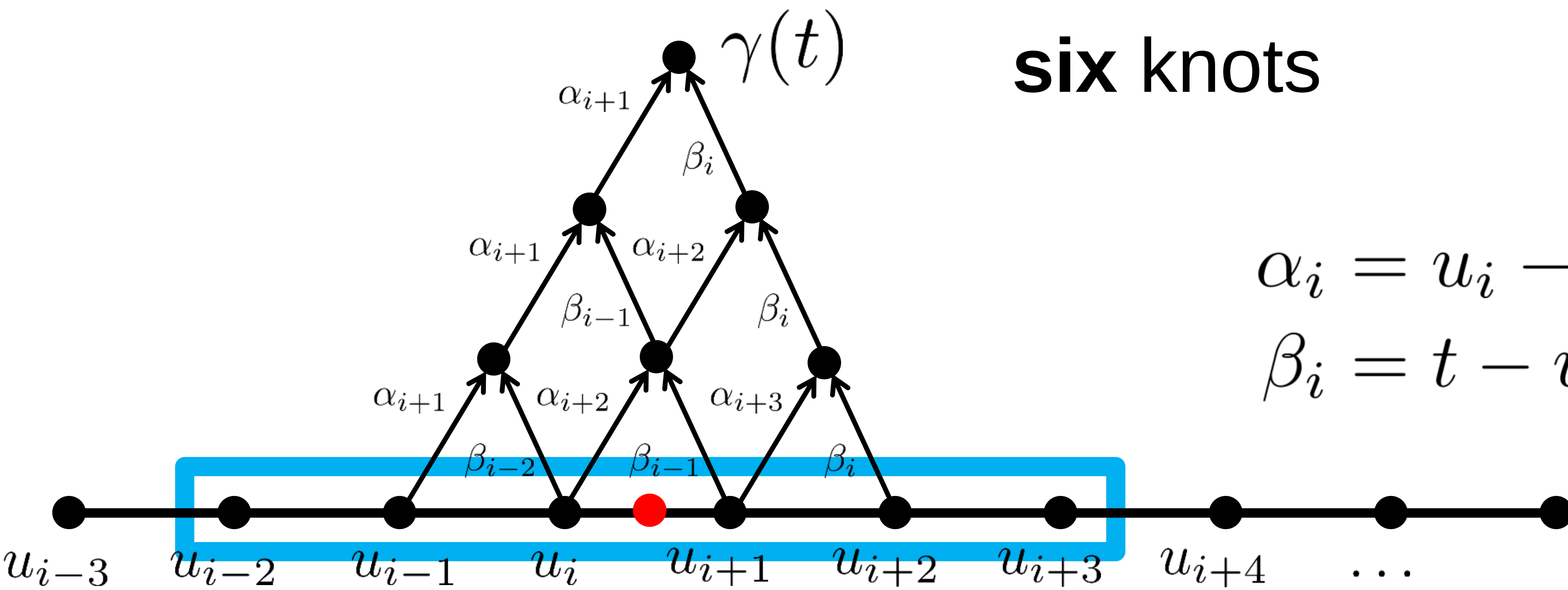
$$\beta_i = t - u_i$$

de Boor's Algorithm

Pyramid algorithm, like de Casteljau

Final answer depends on **four** control pts

six knots

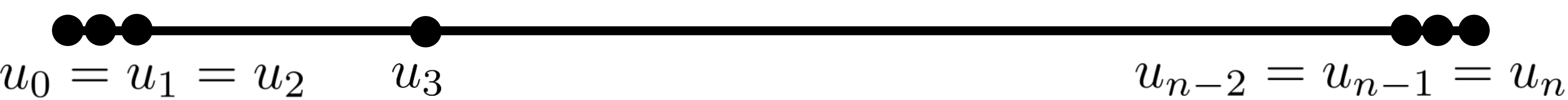


$$\alpha_i = u_i - t$$

$$\beta_i = t - u_i$$

de Boor's Algorithm

Knots triplicated at boundaries



de Boor's Algorithm

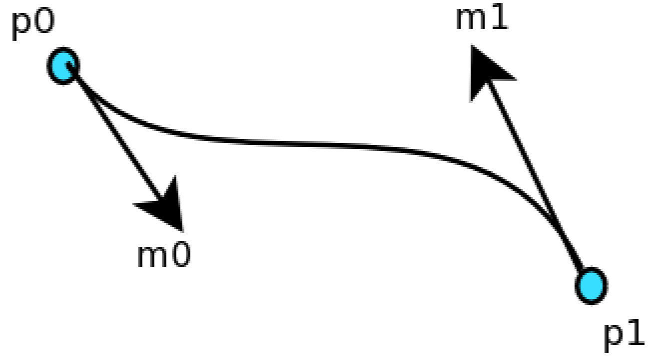
Knots triplicated at boundaries

Higher degree → more pyramid levels
more duplicates at bdry

Other Spline Types

Hermite

- can also specify derivatives at boundary



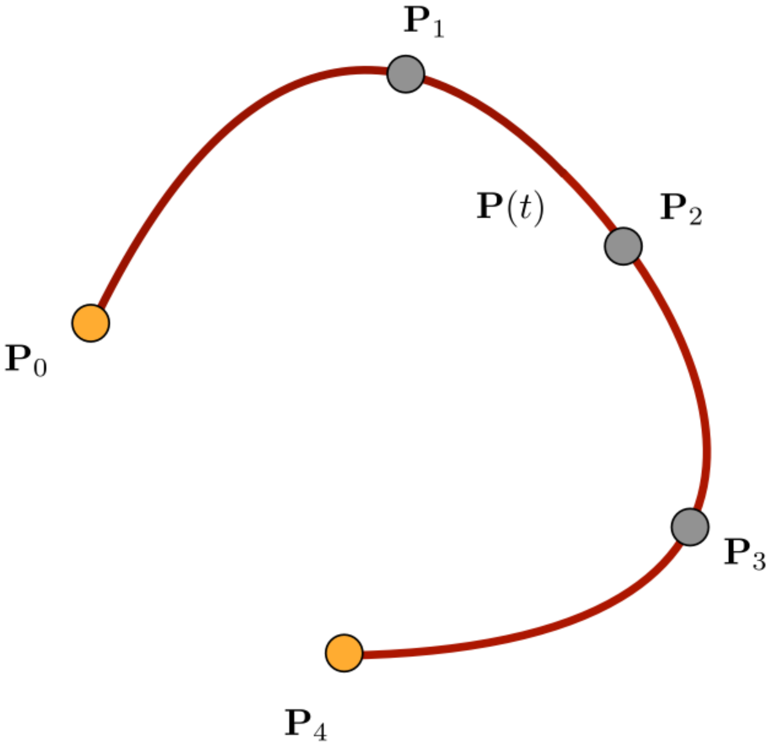
Other Spline Types

Hermite

- can also specify derivatives at boundary

Catmull-Rom

- interpolatory



Spline Keywords

Interpolatory

- spline goes through all control points

Linear

- curve pts linear in **control points**

Degree **n**

- curve pts depend on **n**th power of **t**

Uniform

- knots evenly spaced