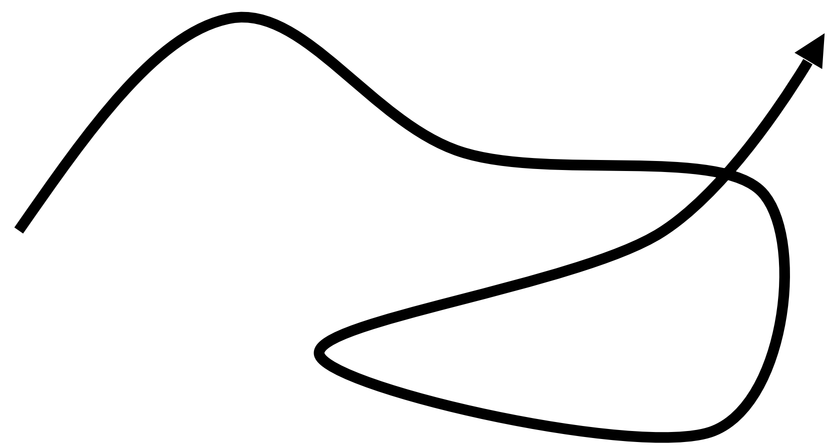
Curves and Splines

Curves in Spaces

Consider some curve with parametric function $\gamma(t)$:



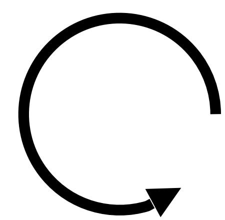
Why might this be a useful thing to know?

Using Parameterized Curves

- Good for:
  - Interpolation in animation
  - Vector-based art (including fonts)
  - Smooth models for physics calculations
- Nice properties:
  - Easy to construct and compute
  - Relatively portable representation

Simpler Curves

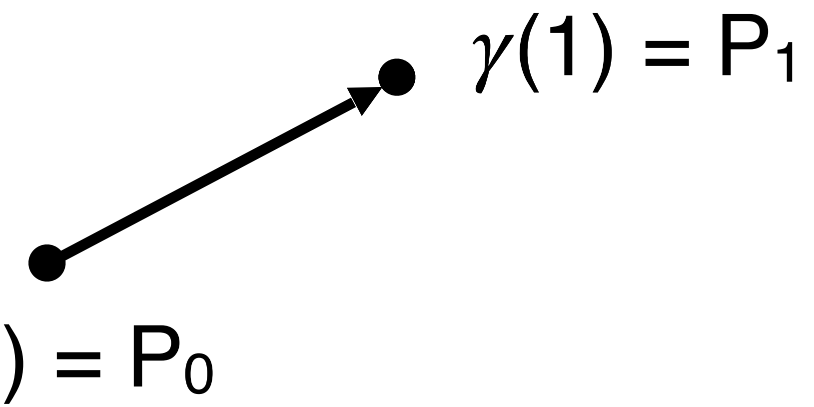
Some formulas more well-known than others: $\gamma(t) = (\cos(t), \sin(t))$



How can we generalize this?

Linear Interpolation

Straight line segment between two points

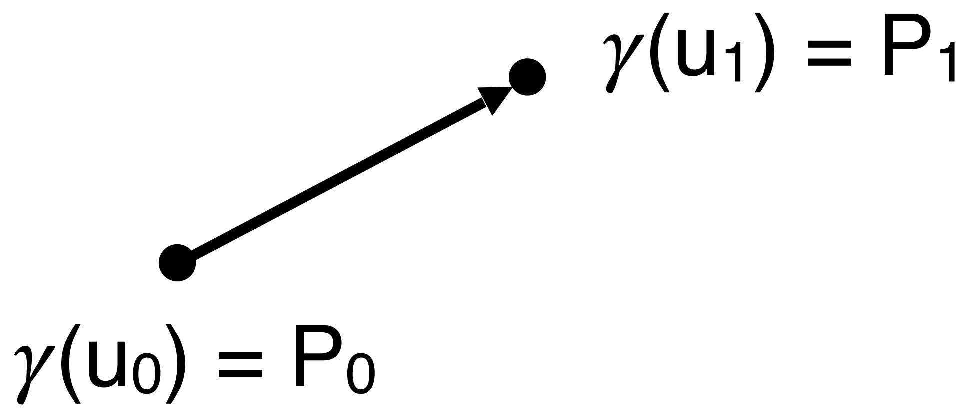


$$\gamma(t) = P_0 + t(P_1 - P_0)$$

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

Using Arbitrary Parameterization

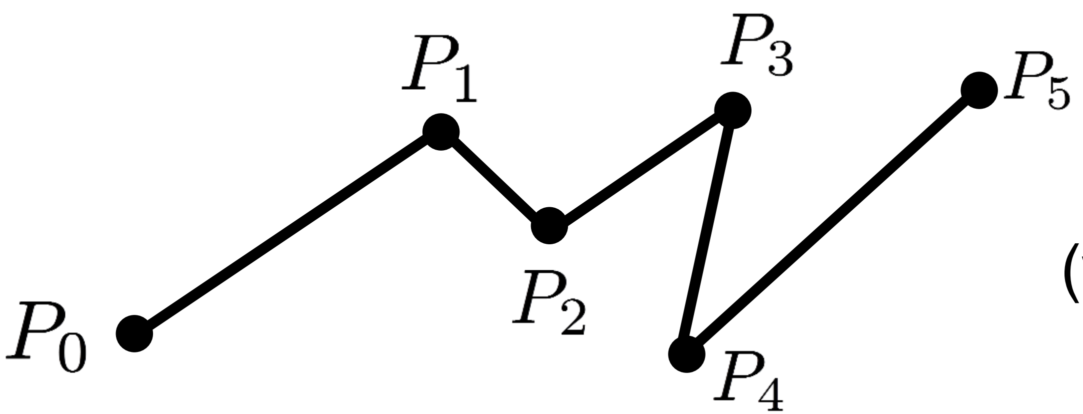
Generalizes to any parameter t…



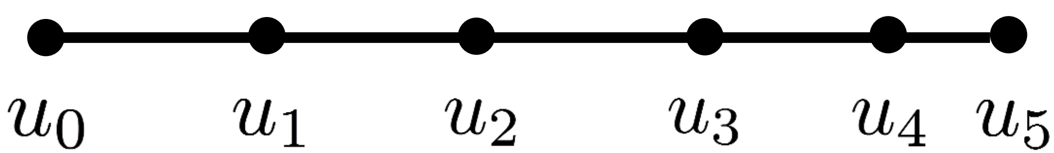
$$\gamma(t) = \frac{u_1 - t}{u_1 - u_0} P_0 + \frac{t - u_0}{u_1 - u_0} P_1$$

Piecewise Linear Interpolation

Straight line segment between point list



points in space
(where curves goes)



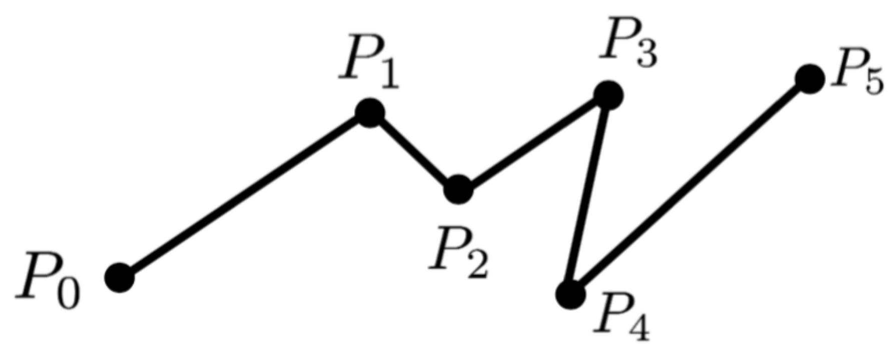
points in parameter space
(**knots**)
(how fast it goes)

In-Class Exercise

Rewrite this parameterization:

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

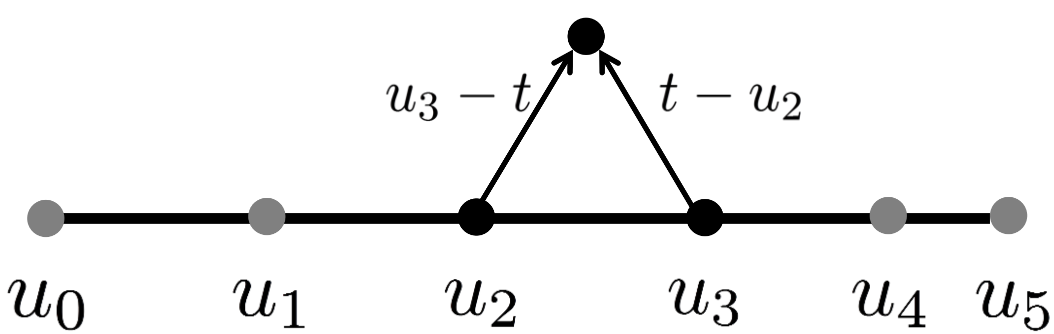
for parameterizing an arbitrary point between P_i and P_{i+1} in a piecewise line segment:



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

A good first approximation:

- Easy to calculate
- Intuitive to understand

Why is this not always sufficient?

Continuity

Smoothness level describes a function's continuity after taking a derivative

C_0: Segments connected at joint

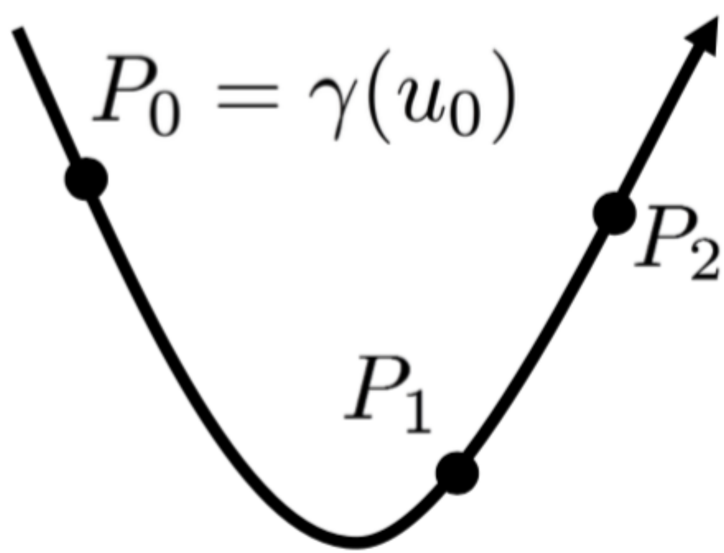
C_1: Segments share 1st derivative at joint

C_2: Segments share 2nd derivative at joint

C_n: Segments share *n*th derivative at joint

Lagrange Interpolation

Given a set of unique data points, possible to construct a polynomial that interpolates between data points



Lagrange Polynomials

Construct polynomial of n-1 degrees from n data points:

$$\sum_{i=0}^{n} \left(\prod_{\substack{0 \le j \le n \\ j \ne i}} \frac{x - x_j}{x_i - x_j} \right) y_i$$

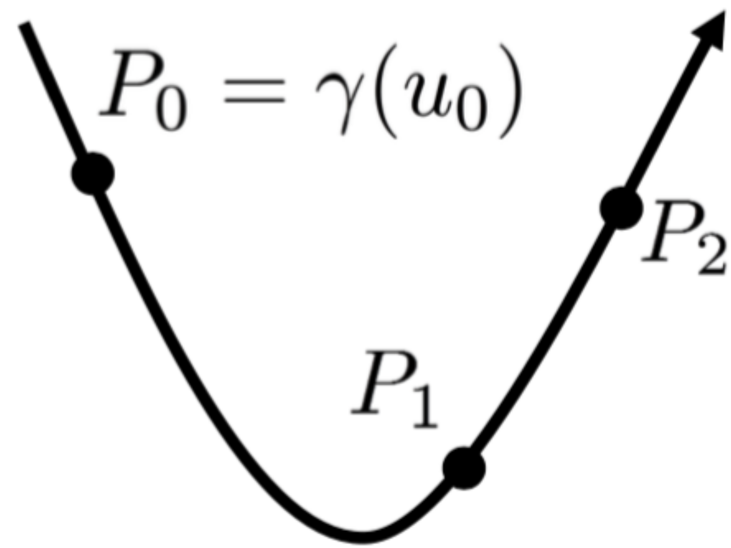
expanded basis polynomial for x_0 from points $\{x_0, x_1, x_2, x_3\}$:

$$L_{3,0}(x) = \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)}$$

Solving as System of Equations

For any point along the curve, there is some polynomial

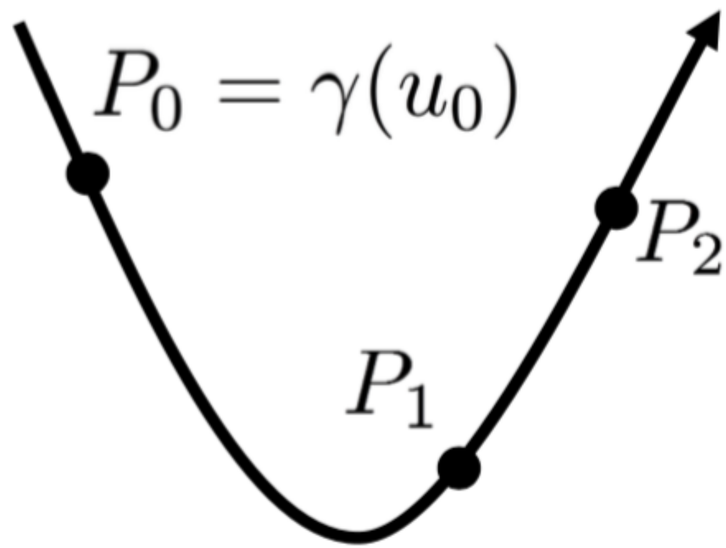
$$\gamma(t) = \left[\begin{array}{c} a_x + b_x t + c_x t^2 + \dots \\ a_y + b_y t + c_y t^2 + \dots \end{array} \right]$$



Lagrange Interpolation

Each coordinate is a linear combination of a power of t

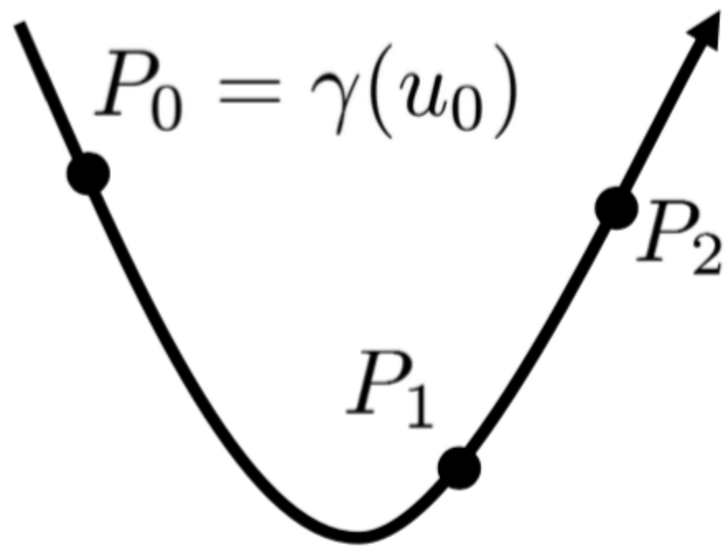
$$\gamma(t) = \begin{bmatrix} a_x & b_x & c_x & \dots \\ a_y & b_y & c_y & \dots \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \end{bmatrix}$$



Lagrange Interpolation

How to solve?

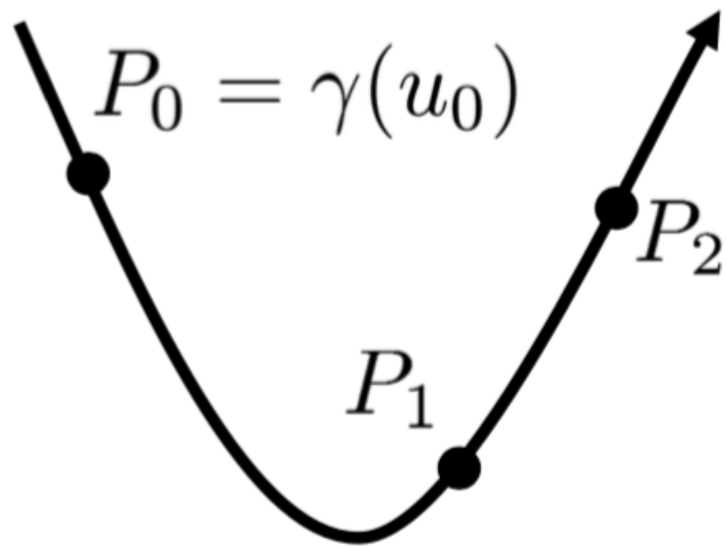
$$\gamma(t) = C_{2\times k} \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \end{bmatrix}_{k\times 1}$$



Lagrange Interpolation

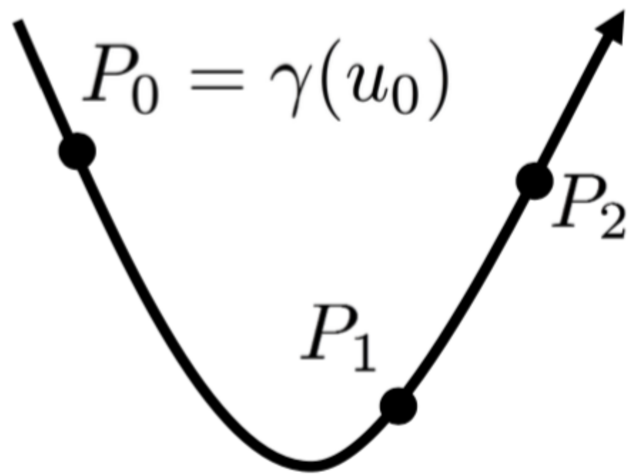
Consider point P_i

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



Lagrange Interpolation

There are *n* known points



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

Vandermonde Matrix

If we use k = n, we get a Vandermonde Matrix

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

Vandermonde Matrix

Inverse of Vandermonde matrix contains coefficients of Lagrange interpolation polynomials

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

Finds coefficients of C

Lagrange vs Vandermonde

Two different methods that solve for the same problem

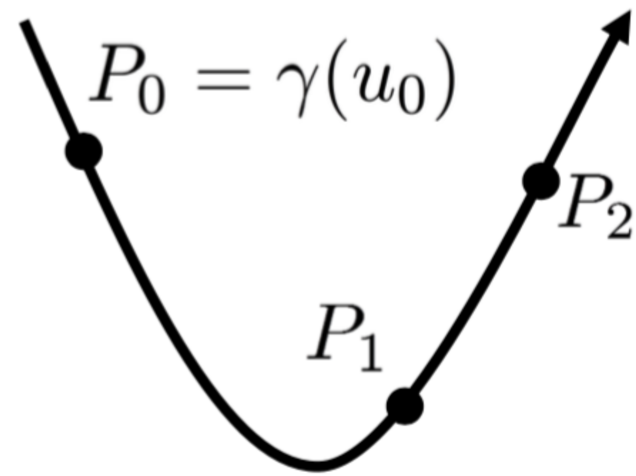
Lagrange interpolation is easier to solve but more involved to use

Vandermonde matrices can be near singular making computation expensive

Lagrange Interpolation

If there are n points, the degree is n-1

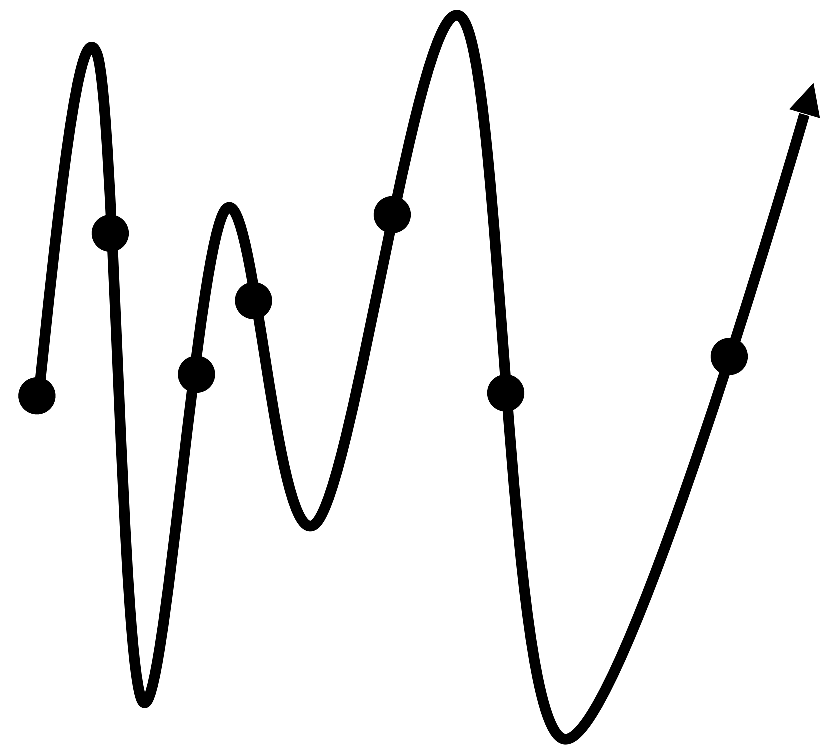
- 2: linear interpolation
- 3: quadratic interpolation



Curves are C^{n-2} smooth

Lagrange Interpolation Problems

- No oscillation control
- Prone to problems of over-fitting
  - Only gets worse with more points
- Must be recalculated if point changes



How can we solve these issues?

Splines

Piecewise polynomial functions

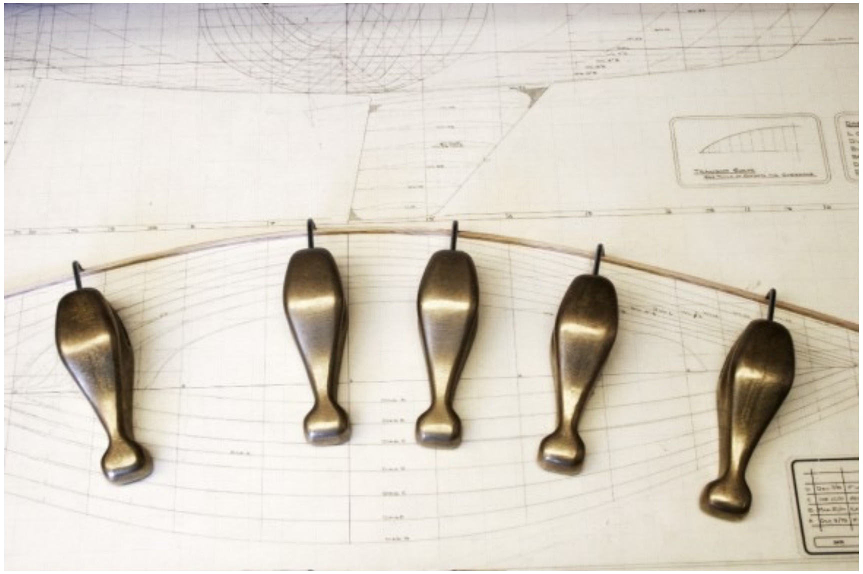
- Allow greater control over specific areas of the curve
- Interpolation more stable than polynomial interpolation
- Guarantees on smoothness at knots

History of Splines

Ship-building tool

Thin strip of wood to model boat’s curves

Weights (ducks or knots) ensure smooth, reproducible curvature



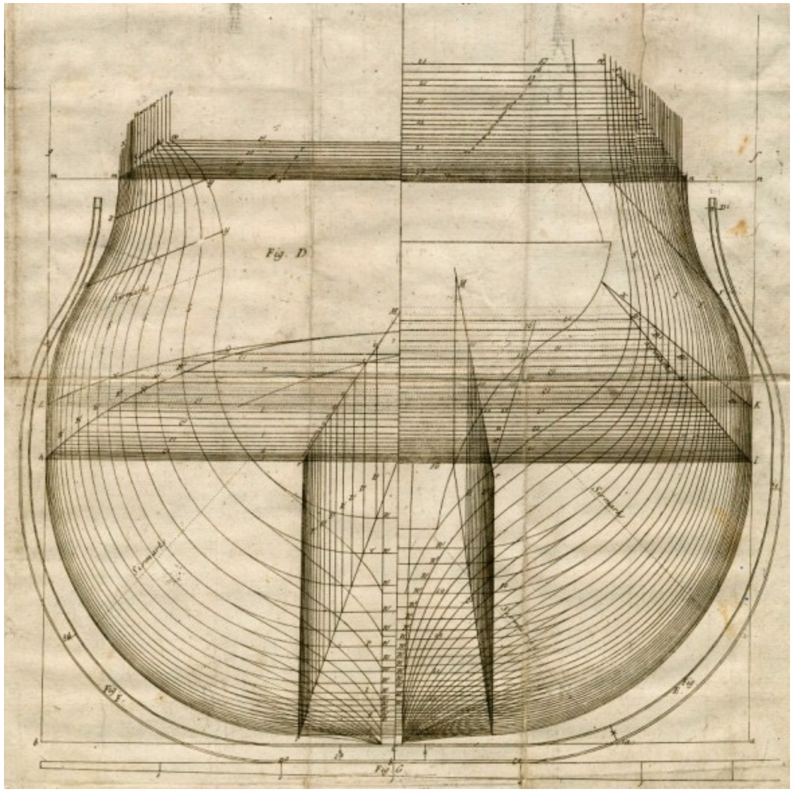
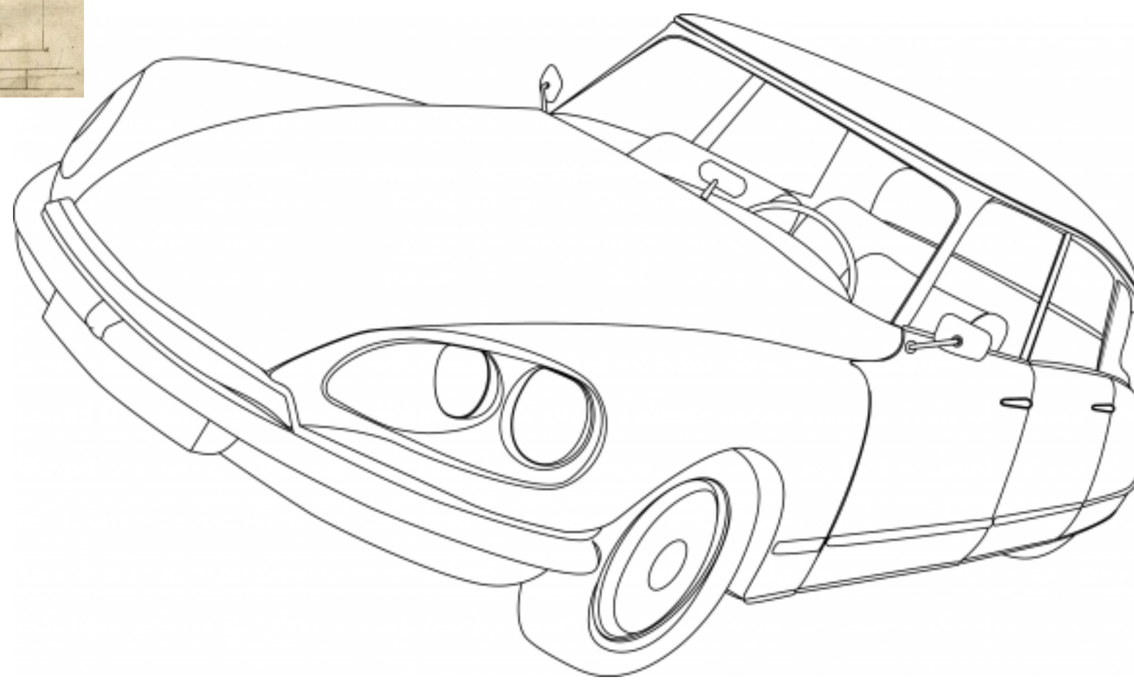


Fig. D



Spline Keywords

Interpolatory

- Spline goes through all control points

Linear

- Curve points linear in **control points**

Degree **n**

- Curve points depend on **n**th power of **t**

Uniform

- Knots evenly spaced

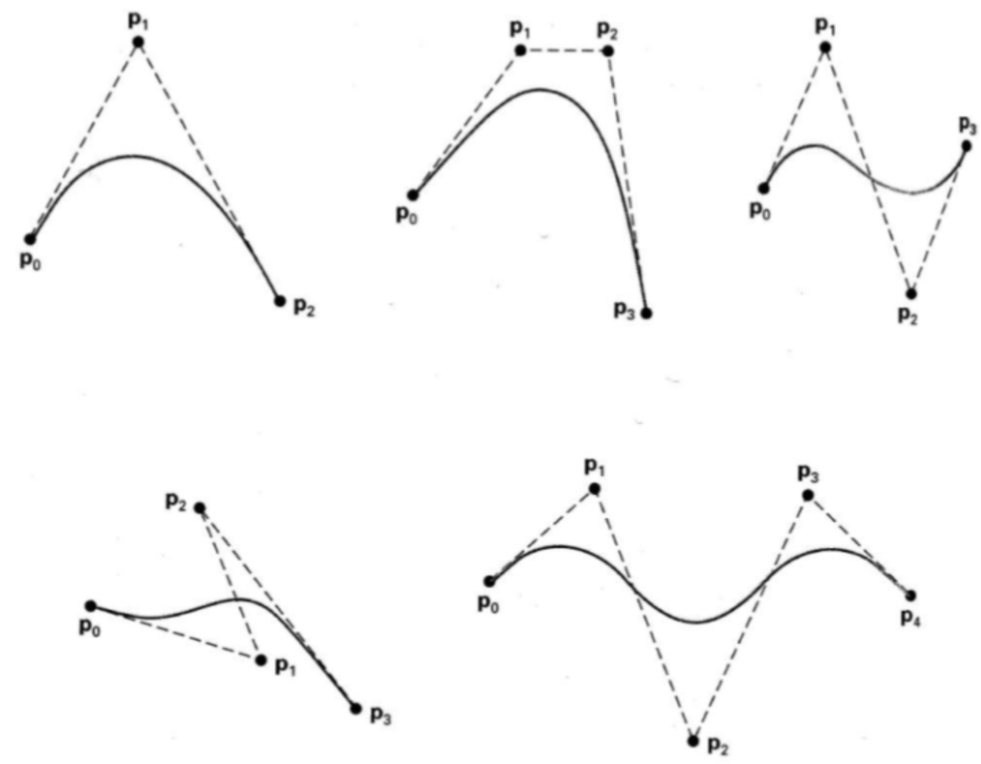
Bézier Curves

Spline building blocks

Polynomial

Control point at each end

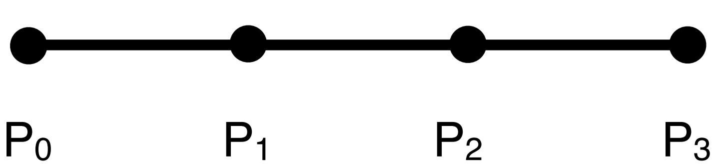
Curve lies in convex hull of control points

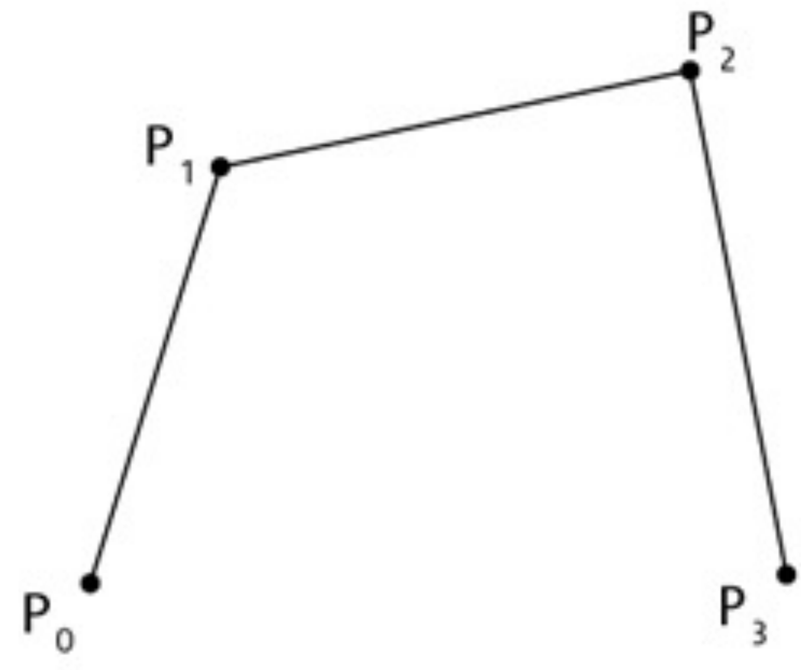


de Casteljau's Algorithm

Main idea: recursive linear interpolation

Start with four points – **control polygon**



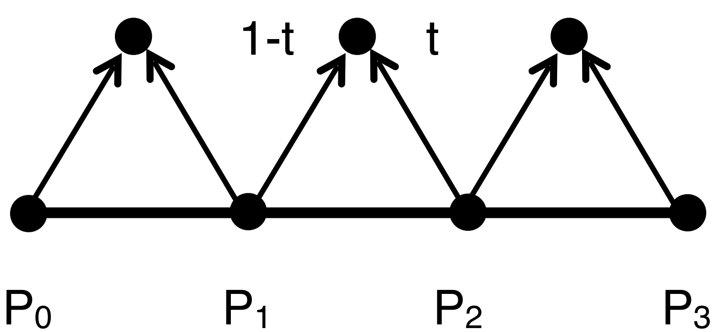


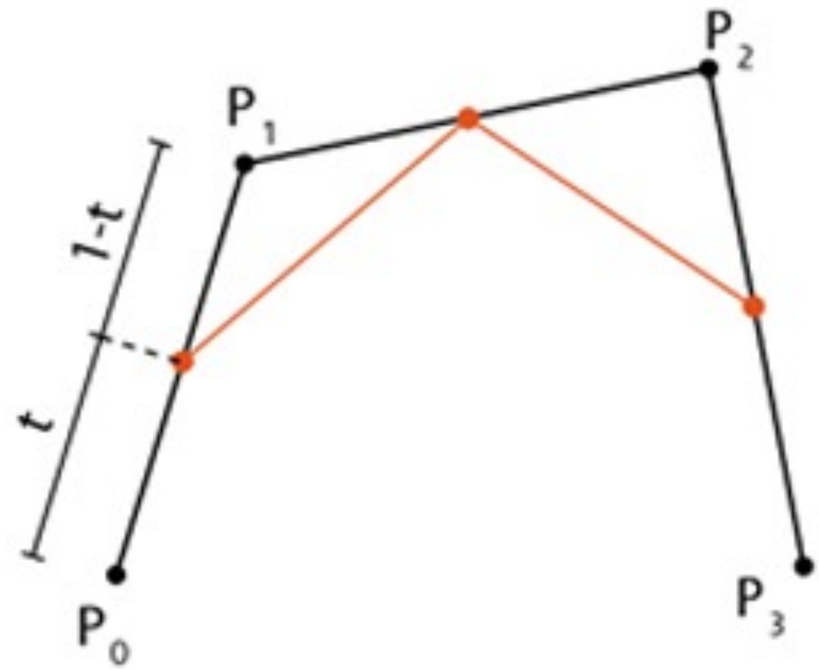
de Casteljau's Algorithm

Main idea: recursive linear interpolation

Start with four points – **control polygon**

Clip corners



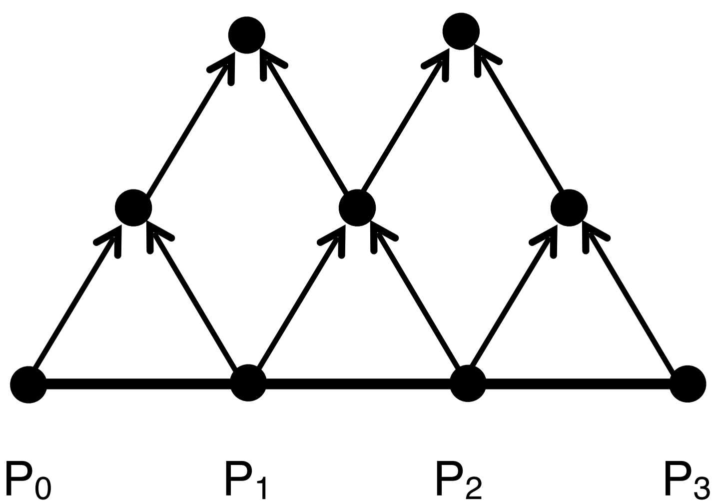


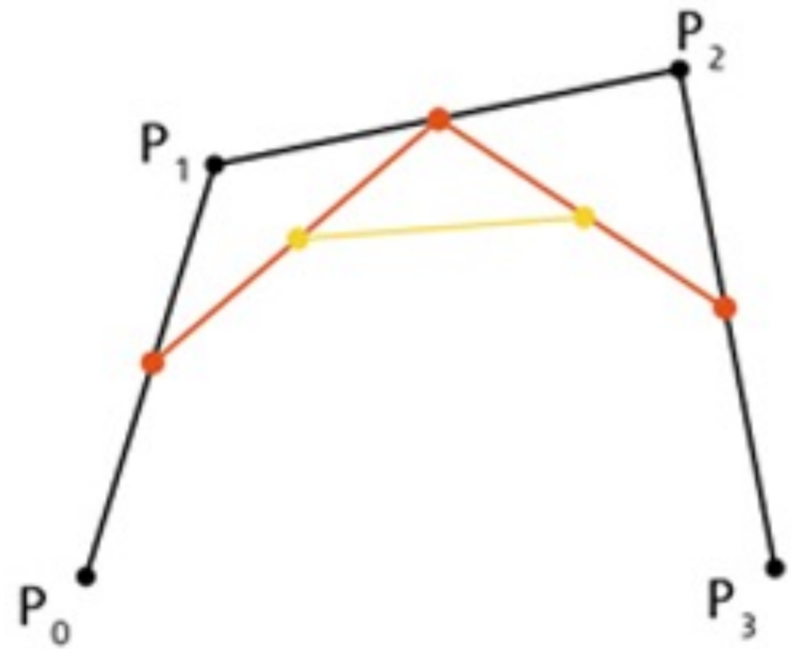
de Casteljau's Algorithm

Main idea: recursive linear interpolation

Start with four points – **control polygon**

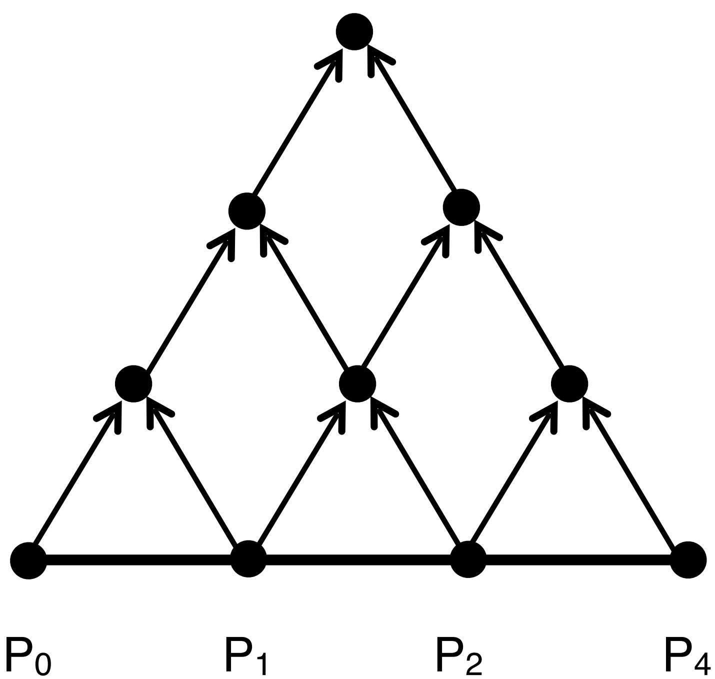
Clip corners

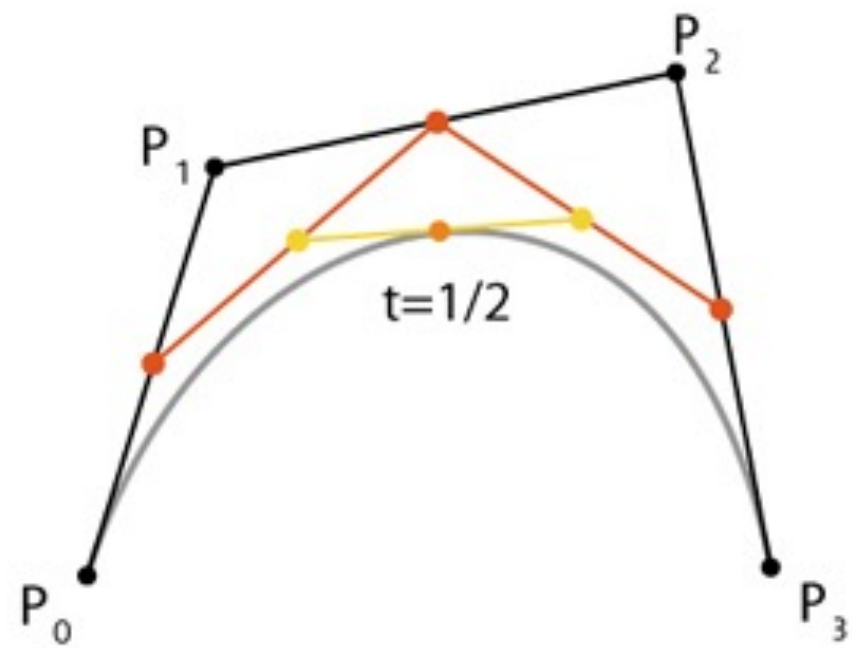




de Casteljau's Algorithm

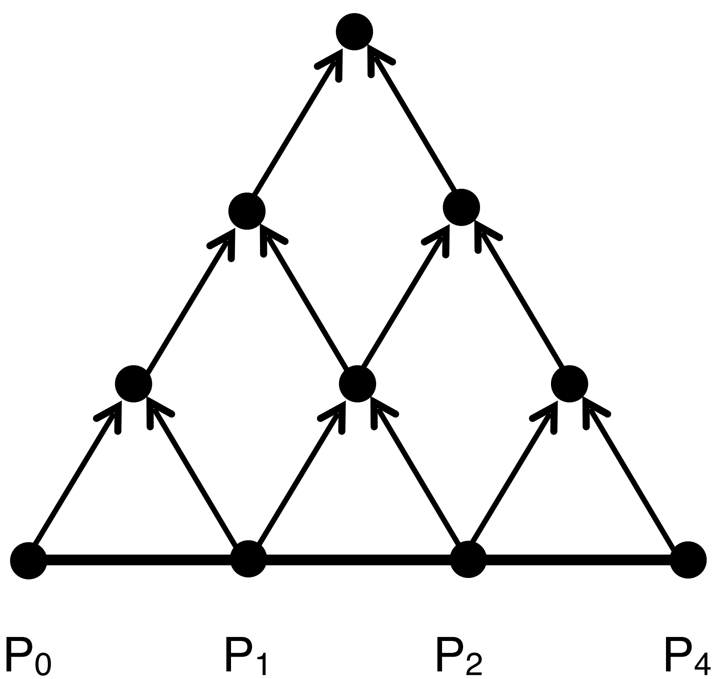
Four control points → cubic Bézier curve

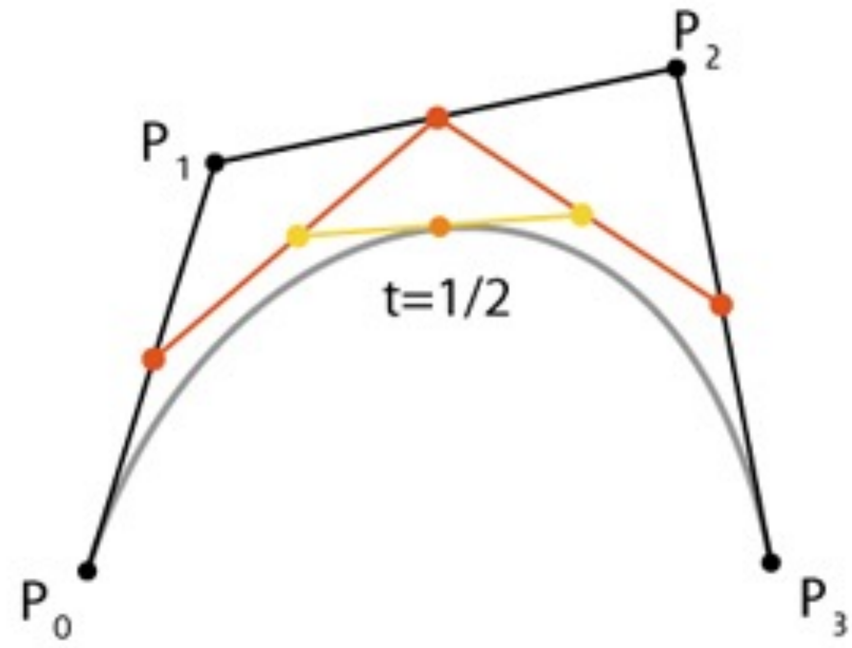


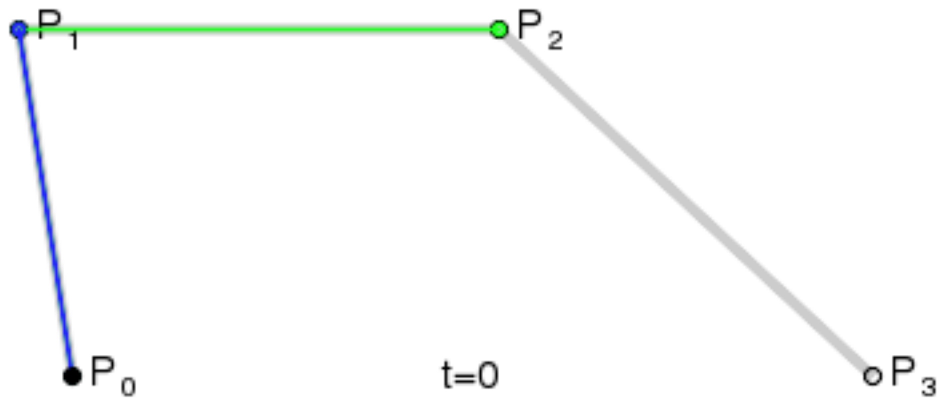


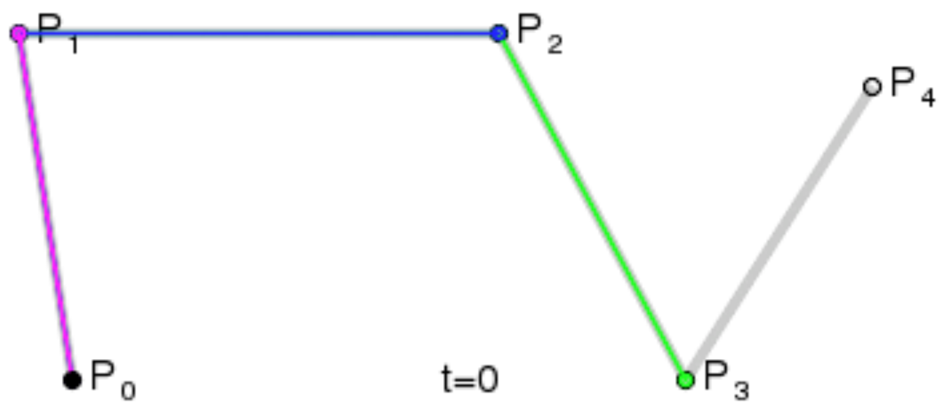
de Casteljau's Algorithm

More control points → smoother curve
(more pyramid levels)









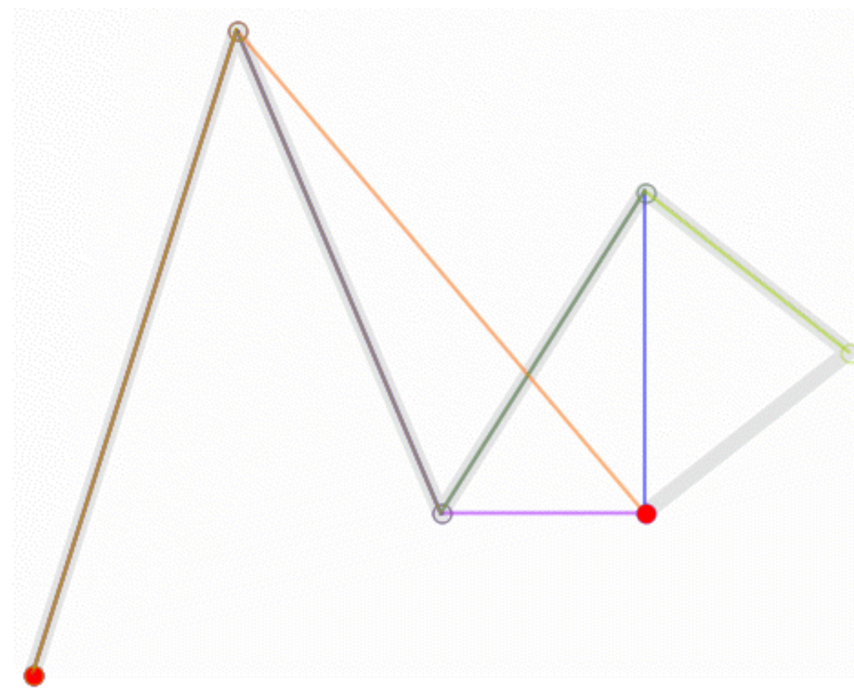
(Wikipedia)

de Casteljau Evaluation

Numerically stable

Slow

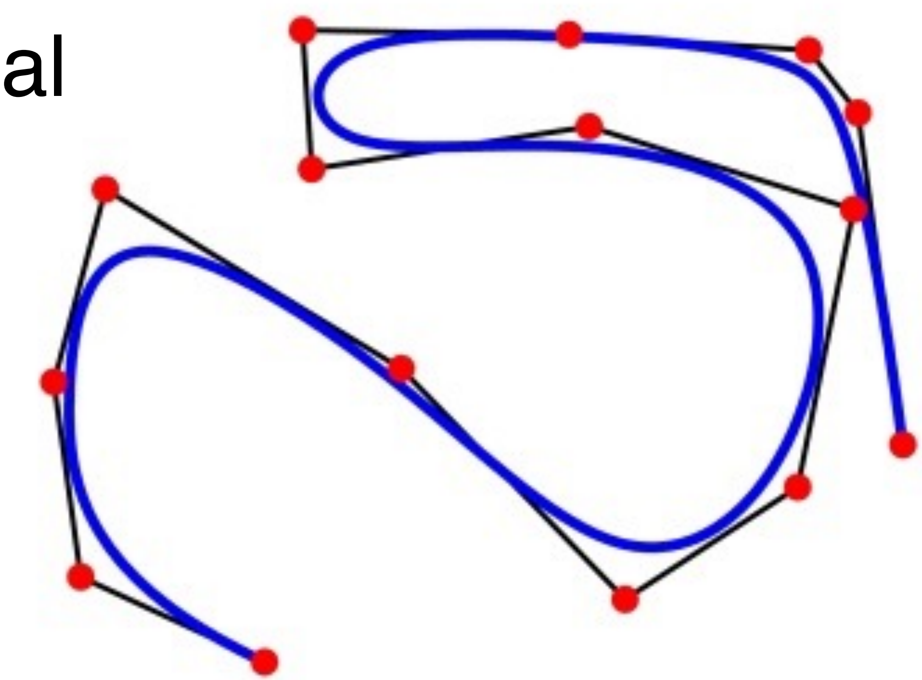
Control points have global influence



B-Splines ("Basis Splines")

Piecewise polynomial

- (cubic common)



Used in Illustrator,
Inkscape, etc

Arbitrary number of control points

- only first and last interpolated

B-Spline Properties

Local support: Polynomials are non-zero in a finite domain (*i* - *n* to *i*) where n is polynomial's degree

Increasing multiplicity of knot decreases number of non-zero basis functions

- If *k* knots at point, at most *n* - *k* + *1* non-zero basis functions at point

de Boor's Algorithm

Pyramid algorithm

Generalization of de Casteljau

Efficient and numerically stable

Allows local influence of control points

Idea: determine curve by inserting a knot
n times (n is degree of polynomial)

Computing de Boor's

If knot has multiplicity of *n*, there is only one non-zero basis function at knot

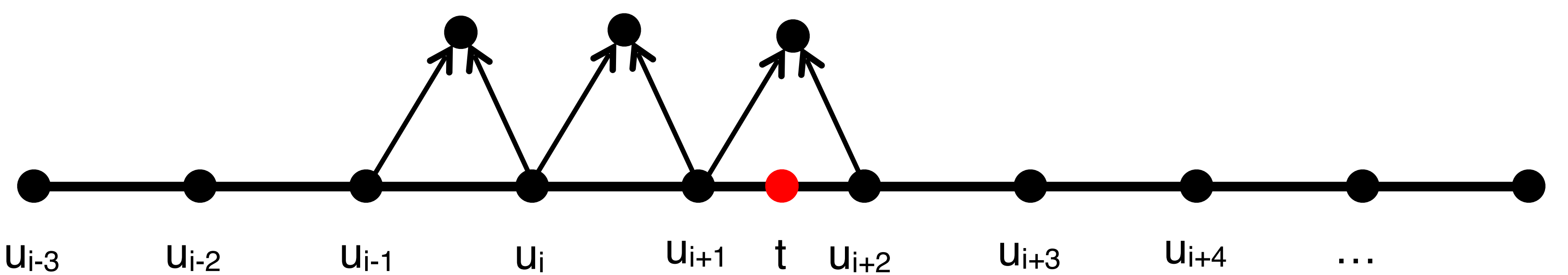
- i.e. the point on the curve is at the control point

If knot is inserted *n* times, final control point calculated from pyramid is point on curve

de Boor's Algorithm

Identify point in space for knot position *t*

- Predetermine spline's degree
- Recursively determine control points (*P*) from local knots (*u*) and previous level of control points

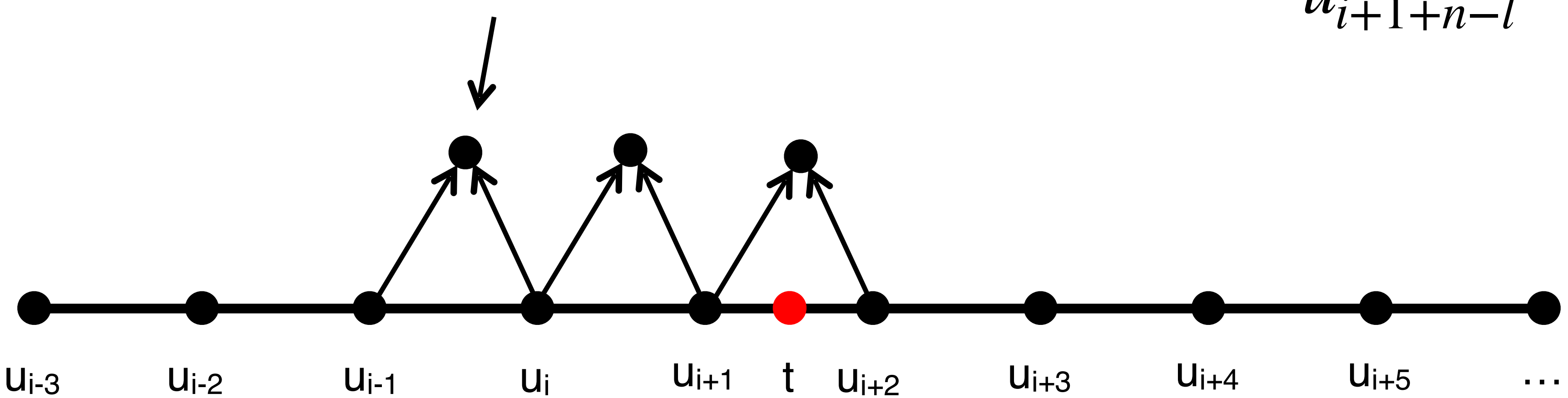


de Boor's Algorithm

Degree 3 spline: $n = 3$

l = pyramid level

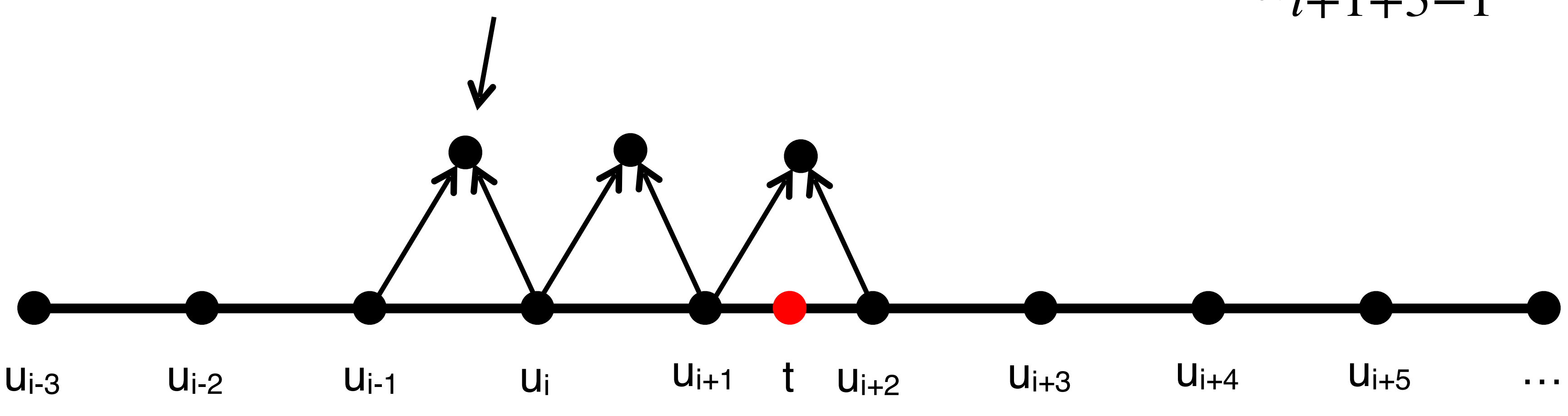
$$P_{i,l}(t) = (1 - \alpha_{i,l})P_{i-1,l-1}(t) + \alpha_{i,l}P_{i,l-1}(t) \qquad \alpha_{i,l} = \frac{t - u_i}{u_{i+1+n-l} - u_i}$$



de Boor's Algorithm: Example

$$P_{i,1}(t) = (1 - \alpha_{i,1})P_{i-1,0}(t) + \alpha_{i,1}P_{i,0}(t)$$

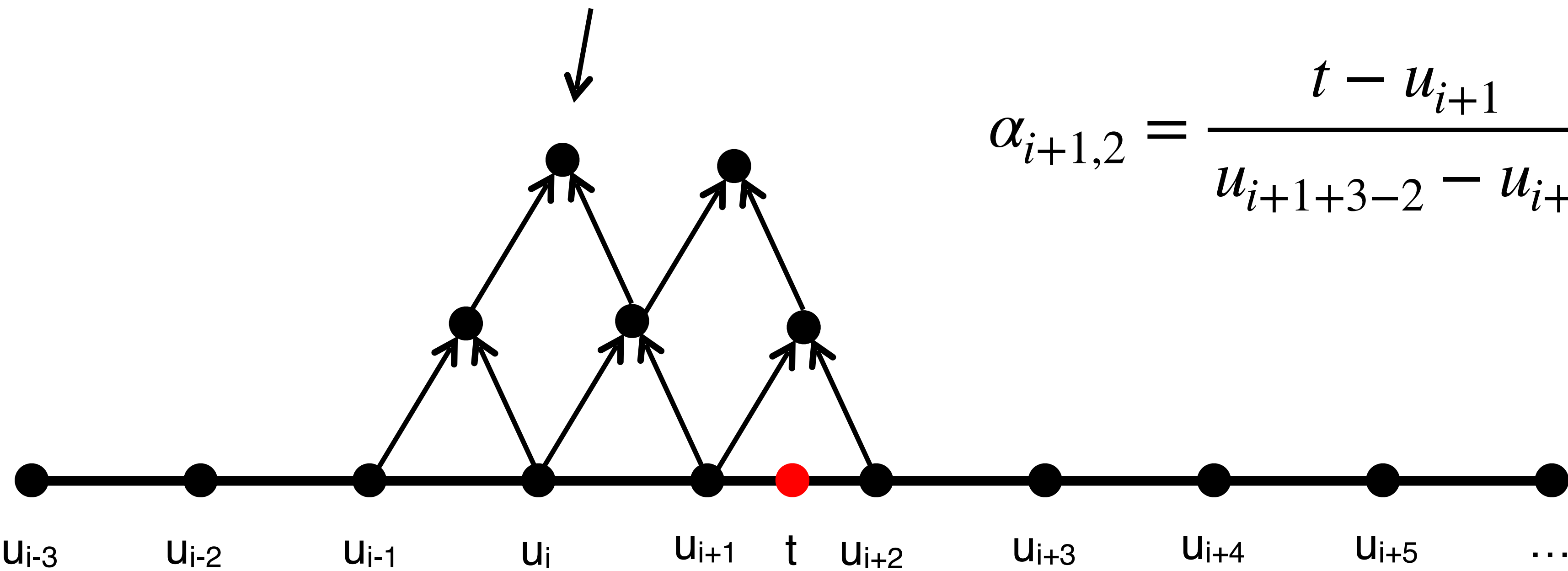
$$\alpha_{i,1} = \frac{t - u_i}{u_{i+1+3-1} - u_i}$$



de Boor’s Algorithm: Example

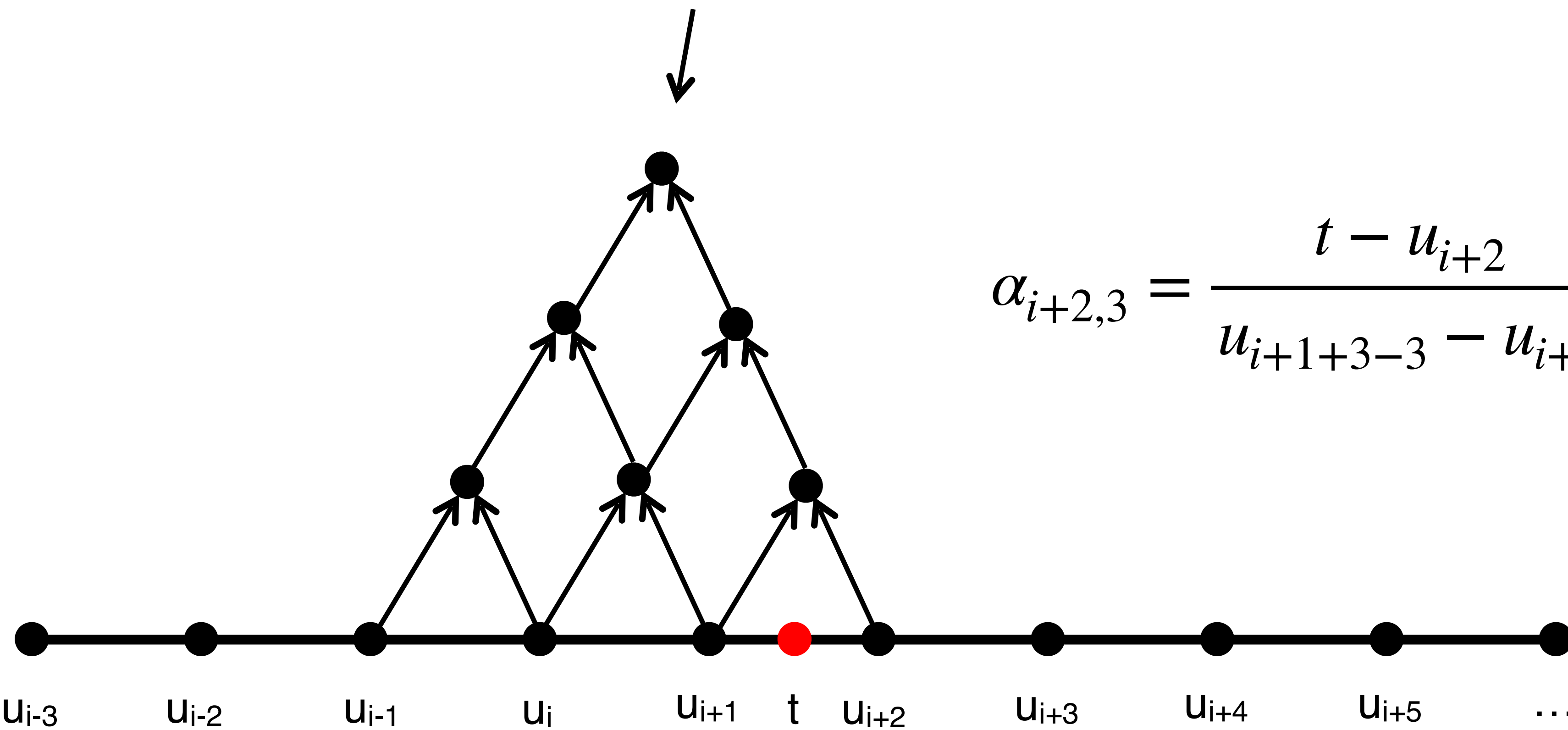
$$P_{i+1,2} = (1 - \alpha_{i+1,2})P_{i,1}(t) + \alpha_{i+1,2}P_{i+1,1}(t)$$

$$\alpha_{i+1,2} = \frac{t - u_{i+1}}{u_{i+1+3-2} - u_{i+1}}$$



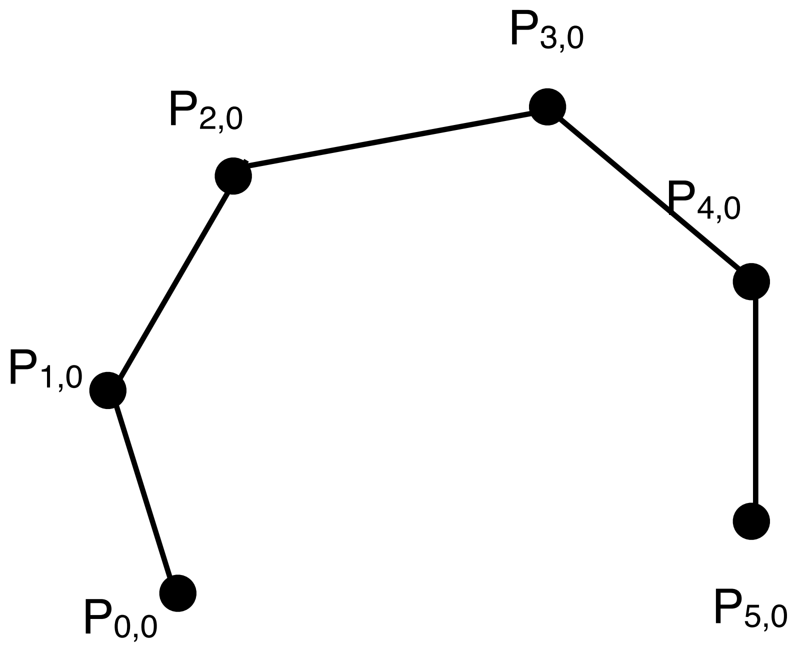
de Boor's Algorithm: Example

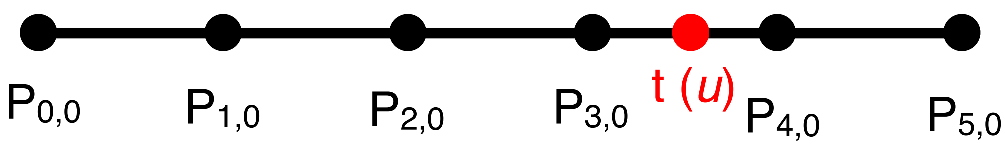
$$P_{i+2,3} = (1 - \alpha_{i+2,3})P_{i+1,2}(t) + \alpha_{i+2,3}P_{i+2,2}(t)$$



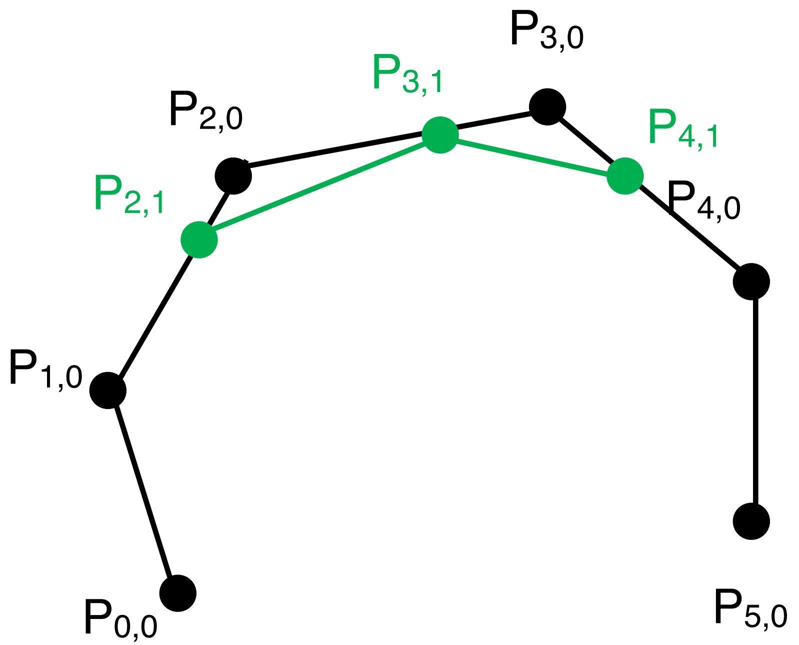
$$\alpha_{i+2,3} = \frac{t - u_{i+2}}{u_{i+1+3-3} - u_{i+2}}$$

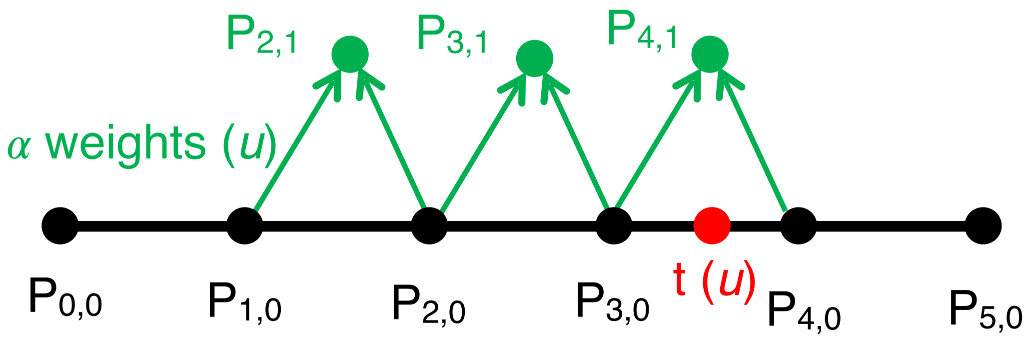
de Boor's Algorithm: Point Space



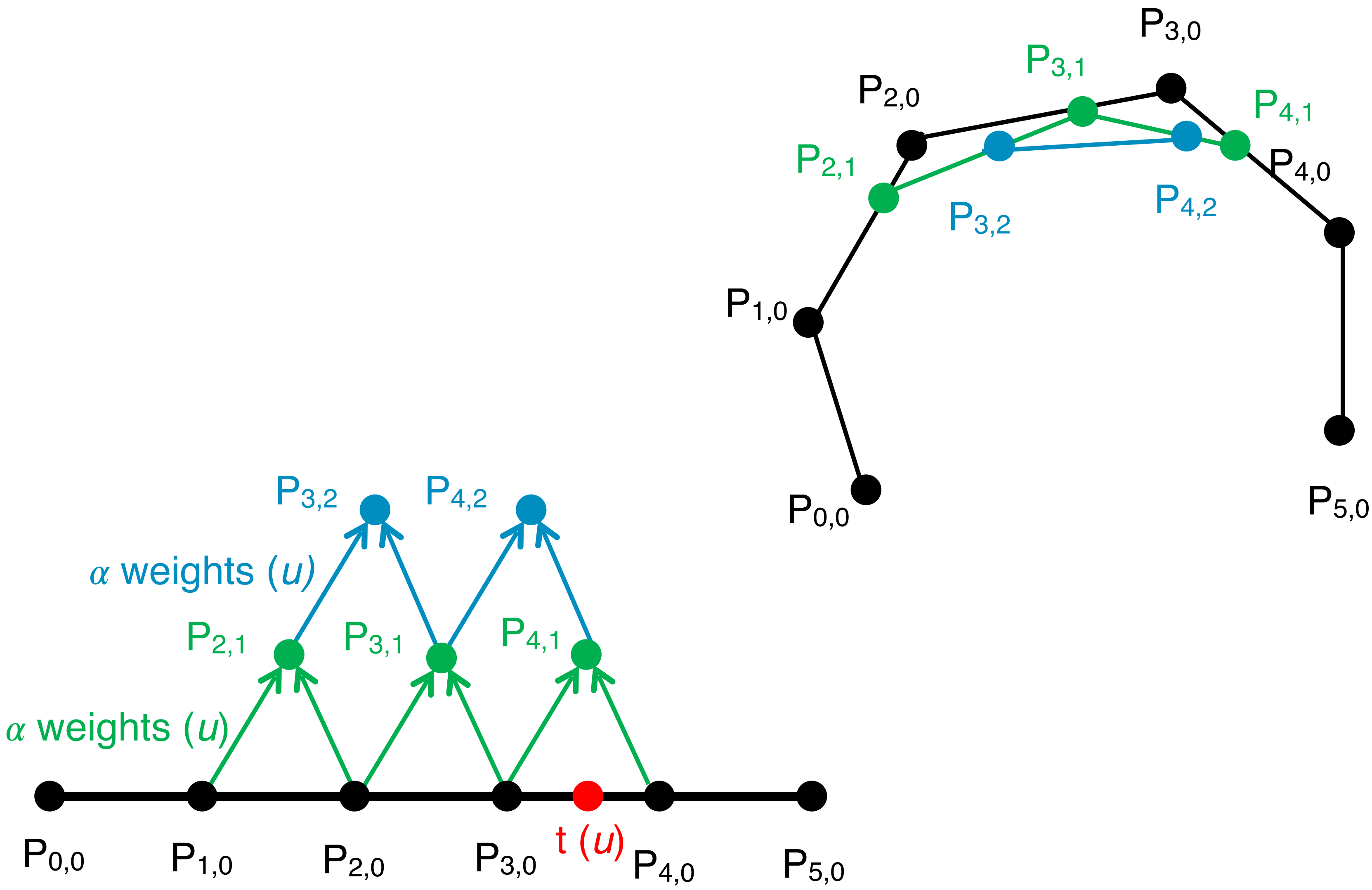


de Boor's Algorithm: Control Points

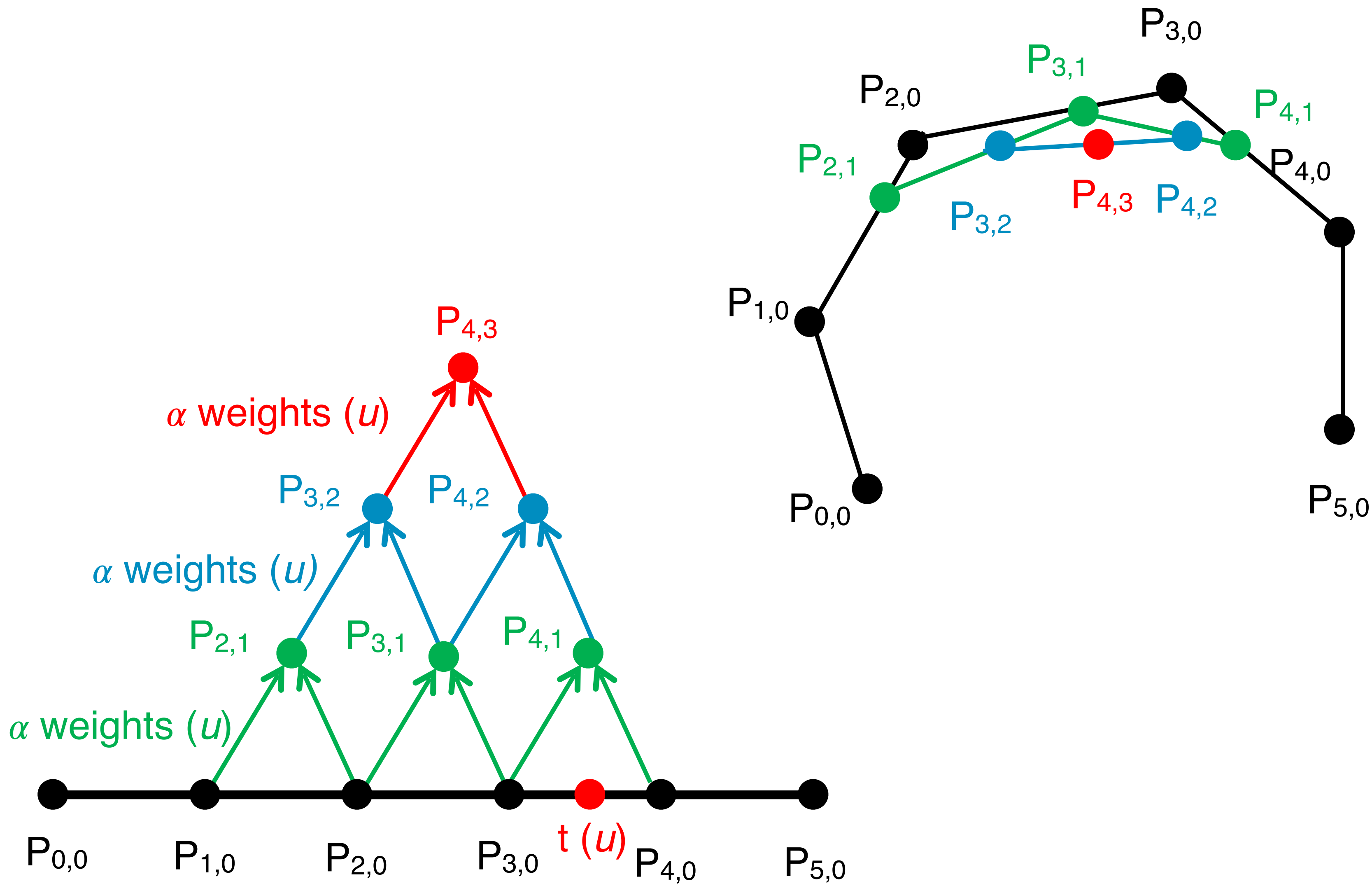




de Boor's Algorithm: Control Points

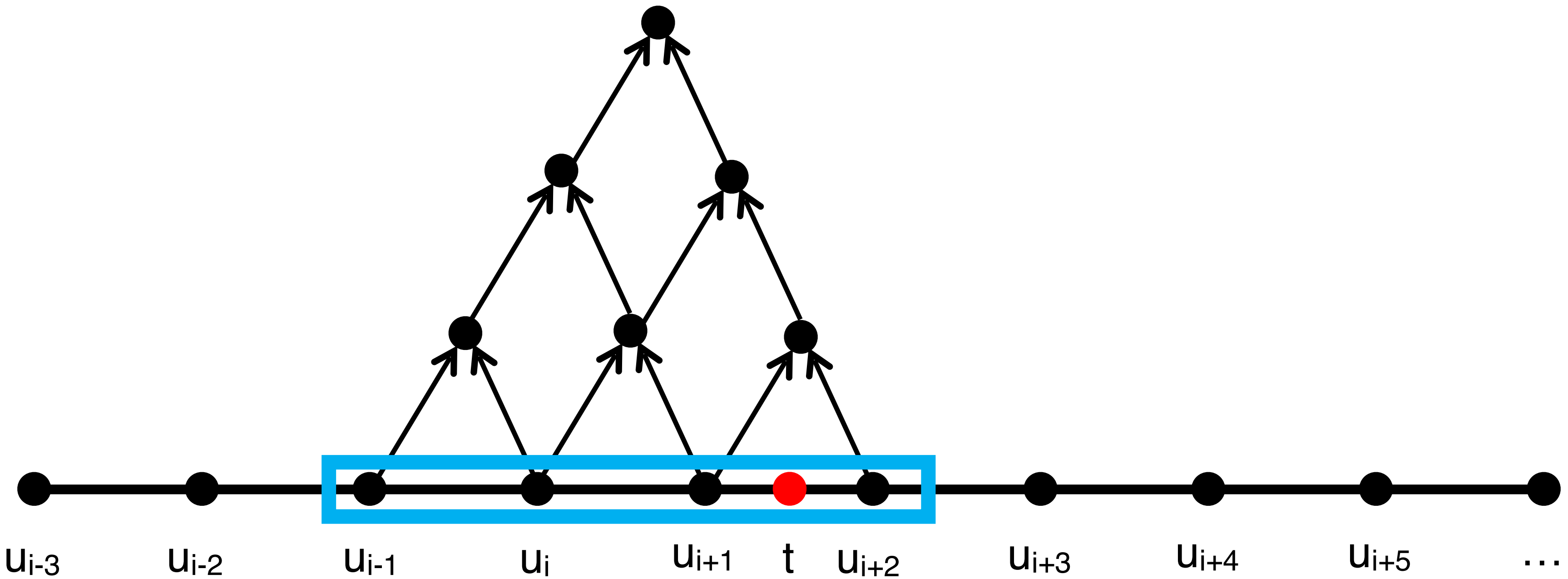


de Boor's Algorithm: Control Points



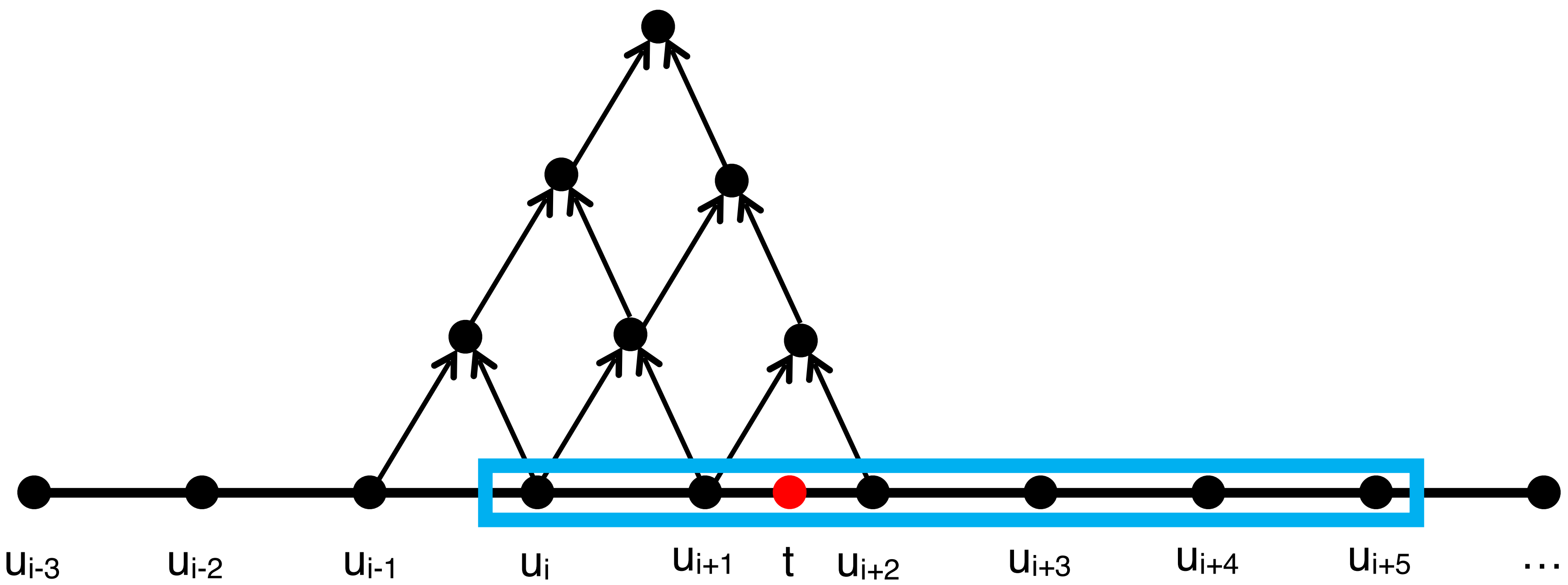
de Boor's Algorithm

Degree 3 spline requires 4 control points…



de Boor's Algorithm

Degree 3 spline requires 4 control points…
And 6 knots…



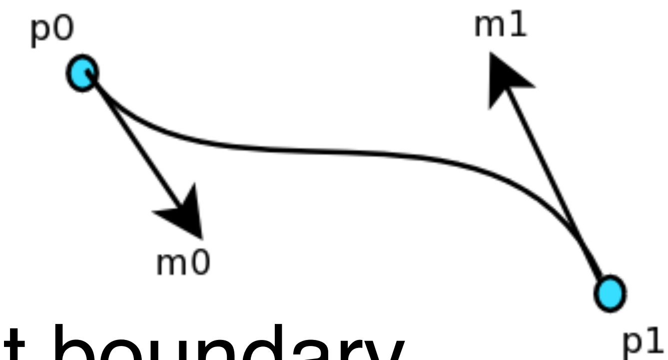
What about the end positions?

de Boor's Algorithm

Knot copied at boundary

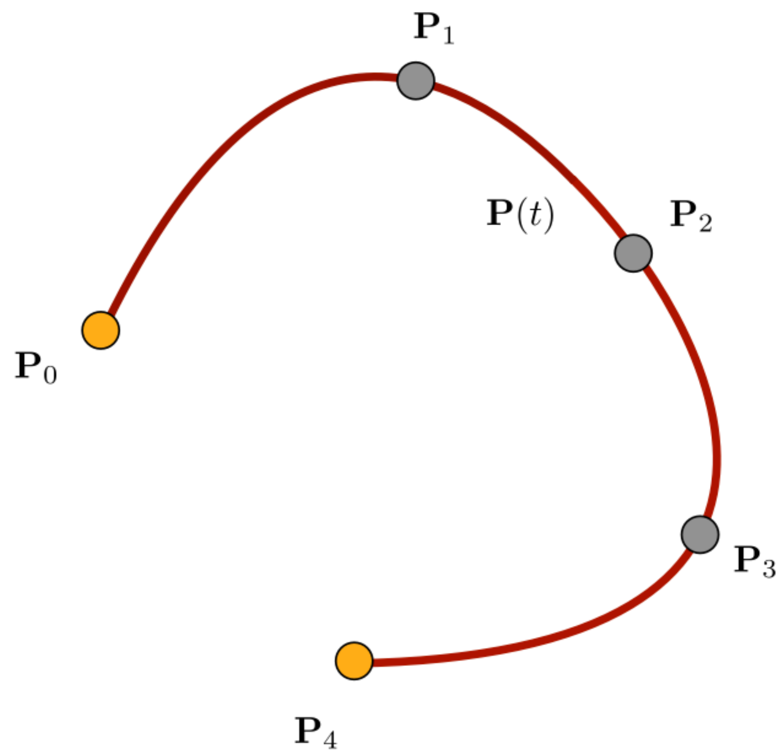
(Higher degree means more levels means more knots)

Other Spline Types



Hermite

- Can specify derivatives at boundary



Catmull-Rom

- Interpolatory

Further Reading

https://cs.uwaterloo.ca/research/tr/1983/CS-83-09.pdf

(History)

http://www.alatown.com/spline/

(de Boor's)

https://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/de-Boor.html

http://www.inf.ed.ac.uk/teaching/courses/cg/d3/deBoor.html