Interpolating Curves

- Intro to curve interpolation & approximation
- Polynomial interpolation
- Bézier curves

Showtime:

Logistics

- Assignment 2 is available
- For assignment questions use the bulletin board or email:
    - csc418tas@cs.toronto.edu
- I'll be away next week, Prof. Singh will be giving the lecture on Wednesday
- Reminder: Midterm held during tutorial time on Monday, Feb. 12
- Covers material from all lectures up to and including this one

Interpolating Curves

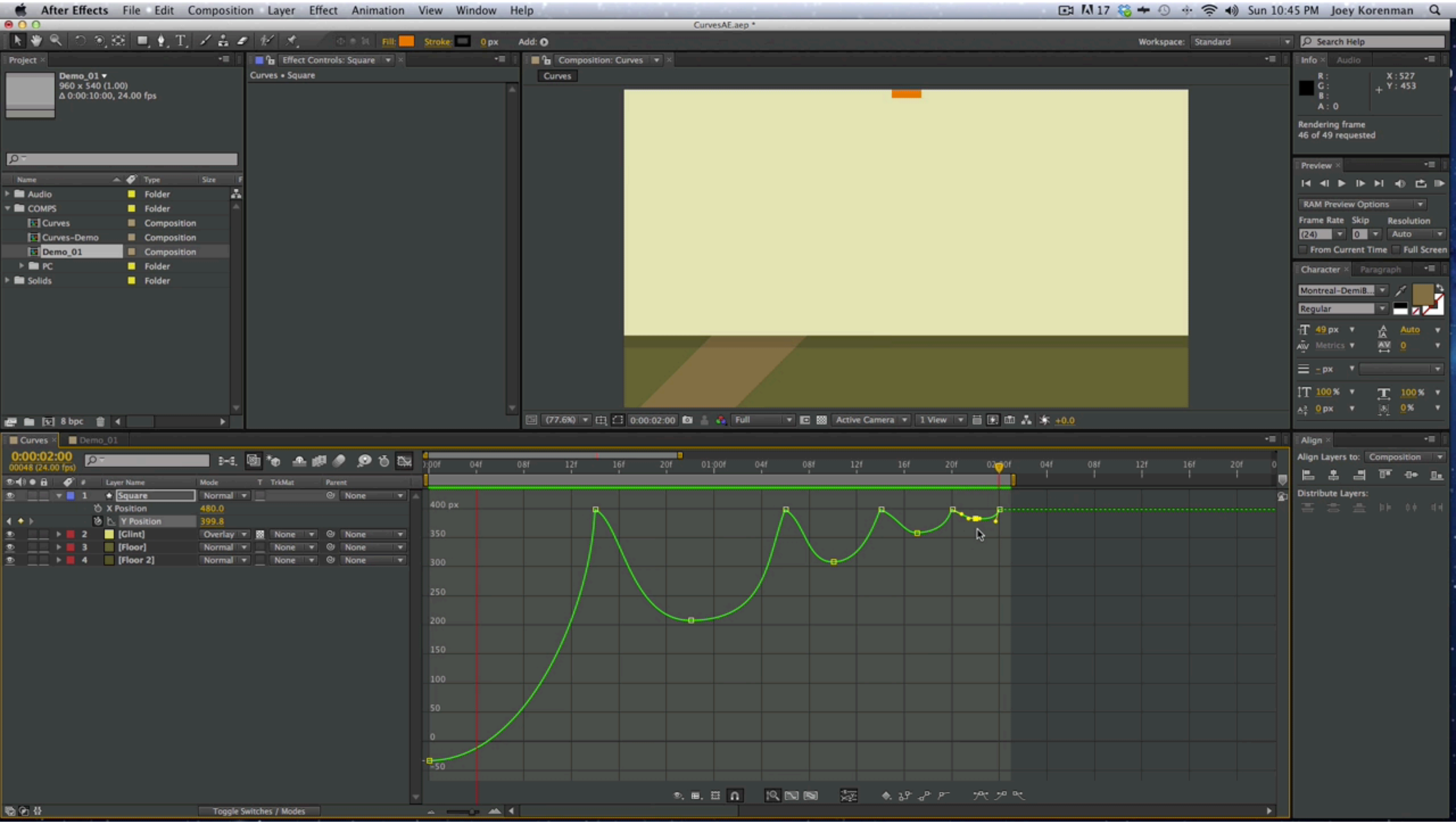
- Intro to curve interpolation & approximation
- Polynomial interpolation
- Bézier curves

Applications

Applications

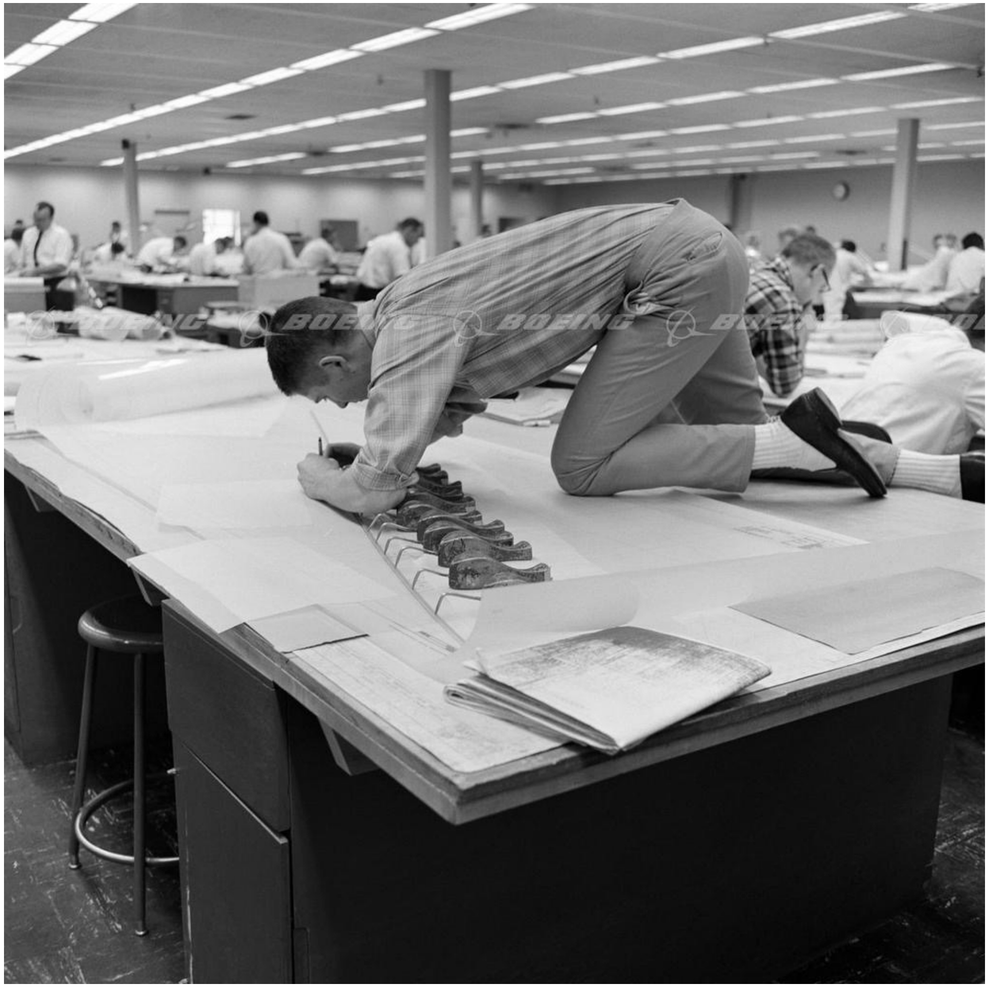
- Specify smooth camera path in scene along spline curve
- Curved smooth bodies and shells (planes, boats, etc)
- Animation curves

Applications



History

- Used to create smoothly varying curves
- Variations in curve achieved by the use of weights (like control points)



Used by engineers in ship building and airplane design before computers were around

Interactive Design of Curves

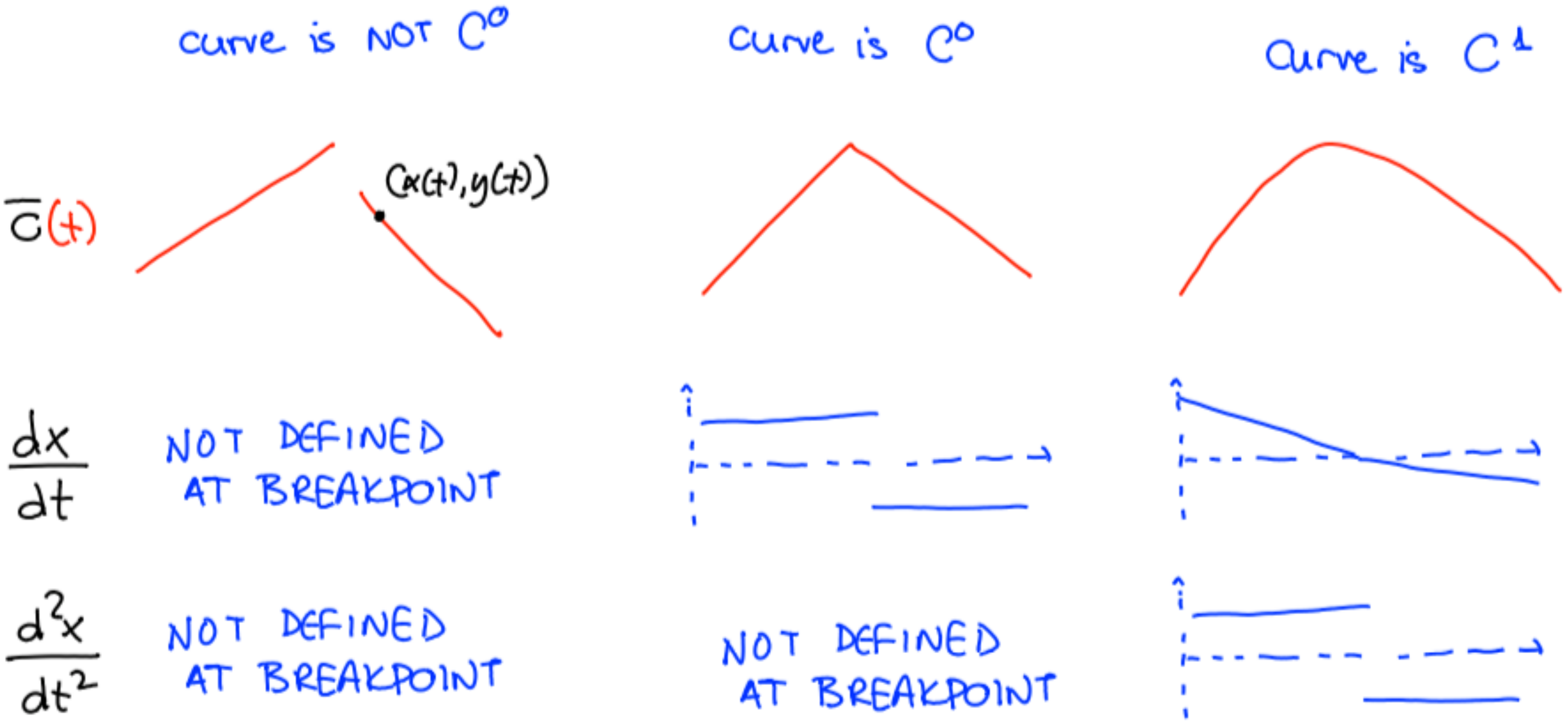
Goal: Expand the capabilities of shapes beyond lines and conics, simple analytic functions and to allow design constraints.

Design Issues:

- Continuity (smoothness)
- Control (local vs. global)
- Interpolation vs. approximation of constraints
- Other geometric properties
  (planarity, tangent/curvature control)
- Efficient analytic representation

C^n continuity

Definition: a function is called C^n if it's nth order derivative is continuous everywhere

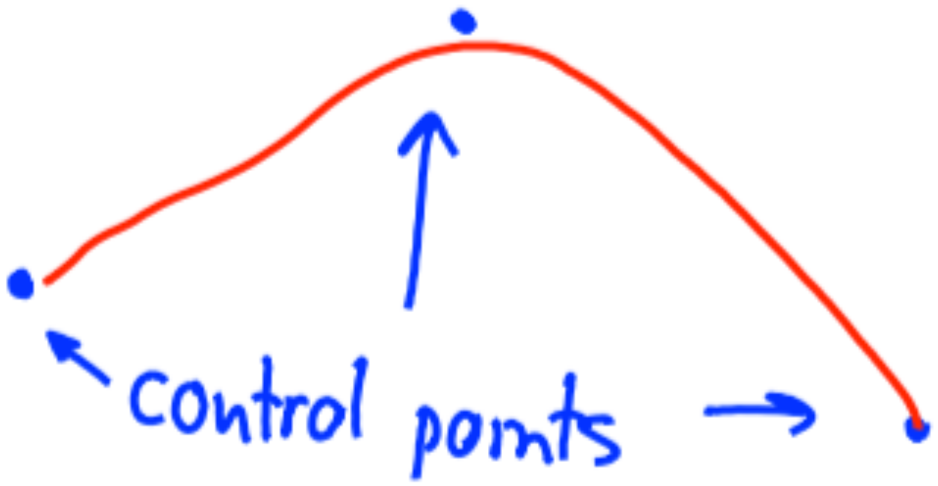


Local vs. Global Control

- Local control changes curve only locally while maintaining some constraints
- Modifying point on curve affects local part of curve or entire curve

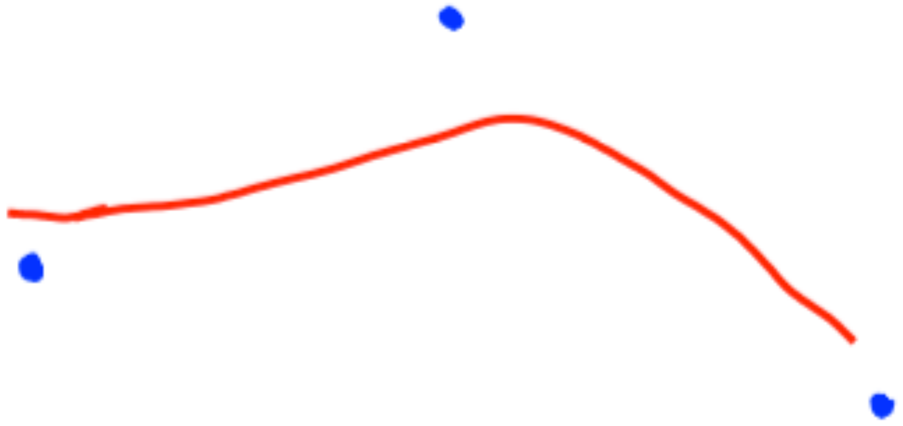
Interpolation vs Approximation

Interpolating splines: pass through all the data points (control points). Example: Hermite splines



Interpolation vs. Approximation

Curve approximates but does not go through all of the control points.



Comes close to them.

Geometric continuity at a joint of two curves

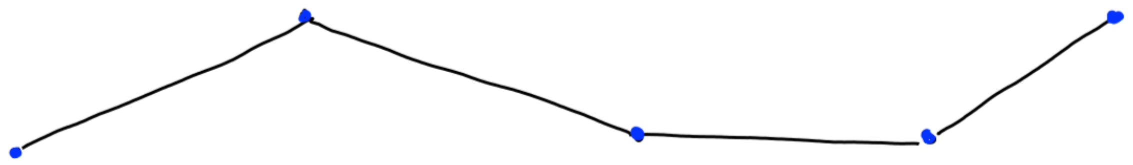
Geometric Continuity
G_0: curves are joined

G_1: first derivatives are proportional at the join point
The curve tangents thus have the same direction,
but not necessarily the same magnitude.
i.e., $C_1'(1) = (a,b,c)$ and $C_2'(0) = (k*a, k*b, k*c)$.

G_2: constant curvature at the join

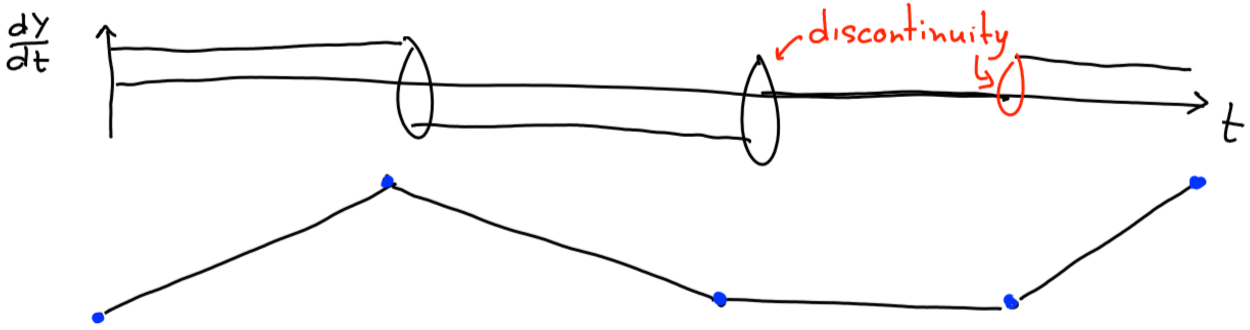
Example: Linear Interpolation

- The simplest possible interpolation technique
- Create a piecewise linear curve that connects the control points



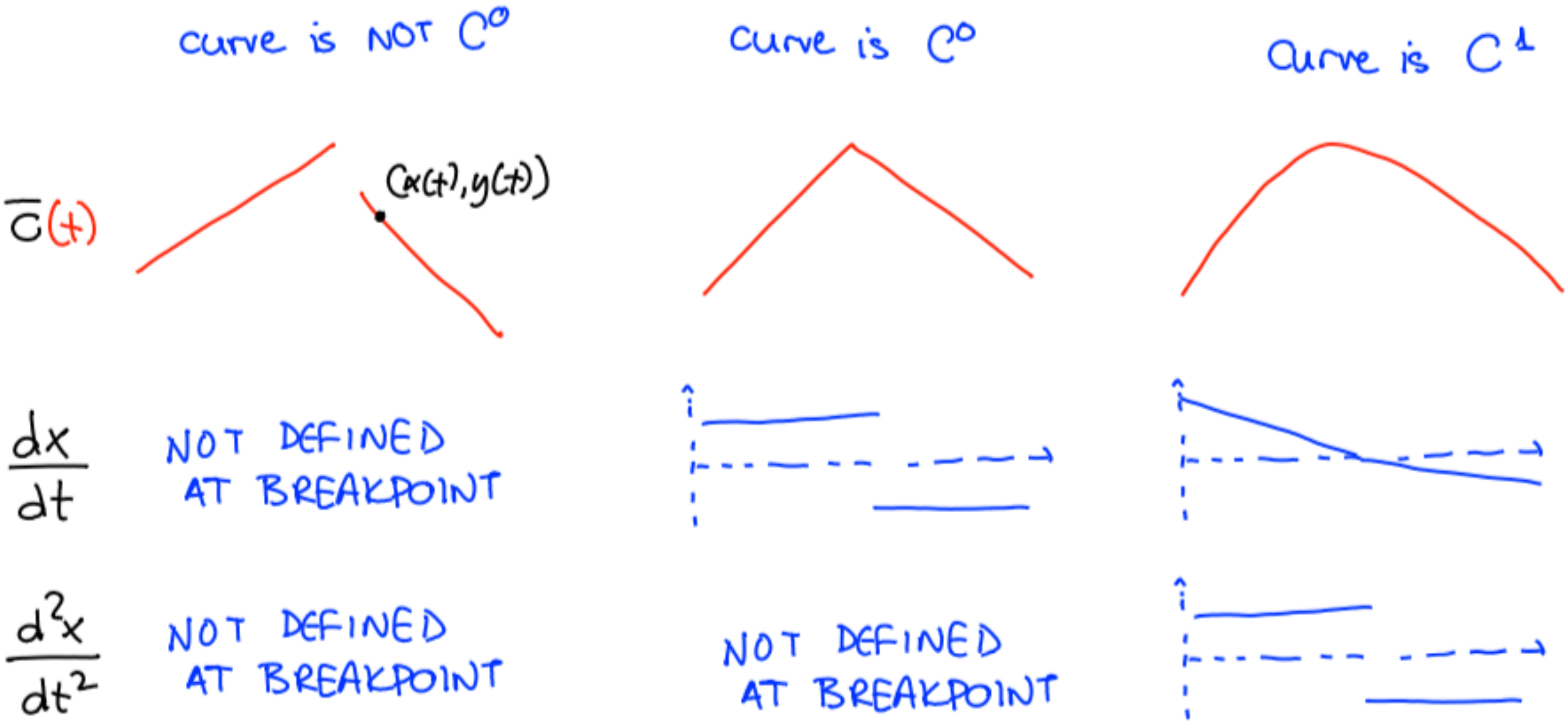
Linear Interpolation

- The simplest possible interpolation technique
- Create a piecewise linear curve that connects the



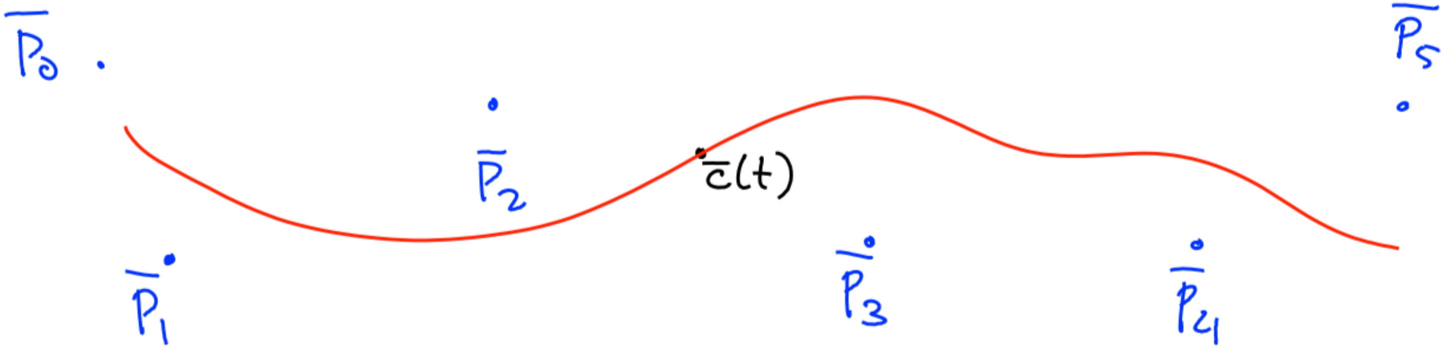
C^n continuity

Definition: a function is called C^n if it's nth order derivative is continuous everywhere



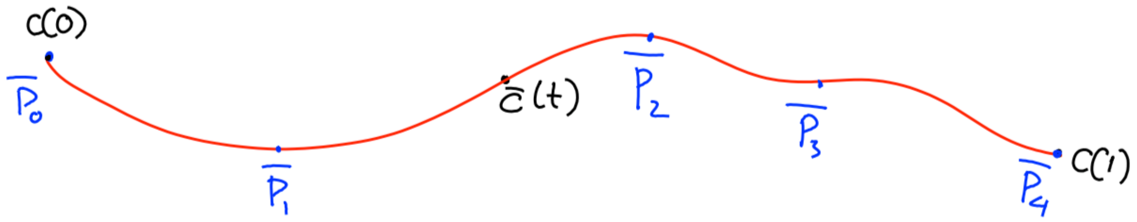
General Problem Statement

- Given N control points, P_i, $i = 0 \ldots n - 1$, $t \in [0, 1]$ (by convention)
- Define a curve c(t) that interpolates / approximates them
- Compute its derivatives (and tangents, normals etc)



Polynomial Interpolation

- Given N control points, P_i, i = 0…n-1, t ∈ [0, 1] (by convention)
  - Define (N-1)-order polynomial x(t), y(t) such that x(i/(N-1)) = $x_{i,}$ y(i/(N-1) = y_i for i = 0, …, N-1
- Compute its derivatives (and tangents, normals etc)



Basic Equations

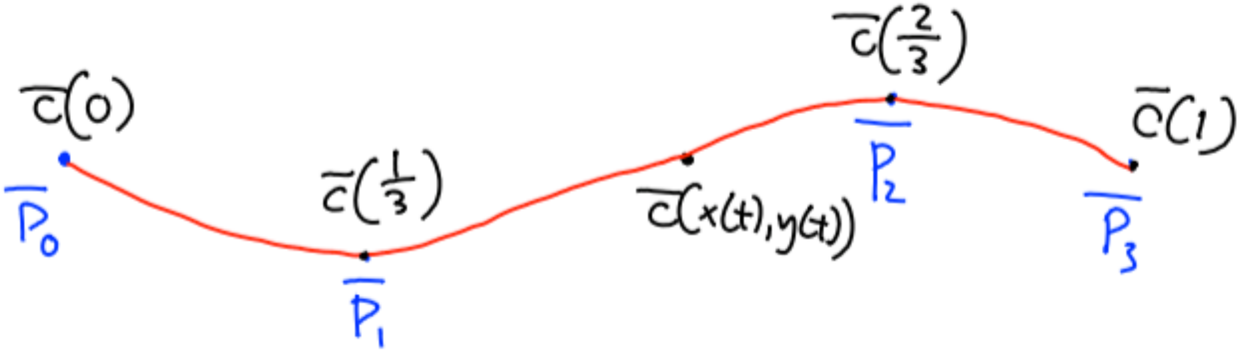
$$\left.\begin{aligned} x(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ y(t) &= b_0 + b_1 t + b_2 t^2 + b_3 t^2 \end{aligned}\right\} \quad \begin{aligned} &\underline{\text{given}}\ \overline{P_1}, \overline{P_2}, \overline{P_3}, \overline{P_4} \\ &\underline{\text{compute}}\ a_i, b_i \end{aligned}$$

Equations for one control point:

$$\begin{aligned} x_1 &= a_0 + a_1 \cdot \tfrac{1}{3} + a_2 \left(\tfrac{1}{3}\right)^2 + a_3 \left(\tfrac{1}{3}\right)^3 \\ y_1 &= b_0 + b_1 \cdot \tfrac{1}{3} + b_2 \left(\tfrac{1}{3}\right)^2 + b_3 \left(\tfrac{1}{3}\right)^3 \end{aligned}$$

Equations in matrix form:

$$\begin{bmatrix} x_1 & y_1 \end{bmatrix} = \begin{bmatrix} 1 & \tfrac{1}{3} & \left(\tfrac{1}{3}\right)^2 & \left(\tfrac{1}{3}\right)^3 \end{bmatrix} \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix}$$



Computing Coeffs

$$\left.\begin{aligned} x(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ y(t) &= b_0 + b_1 t + b_2 t^2 + b_3 t^2 \end{aligned}\right\}$$

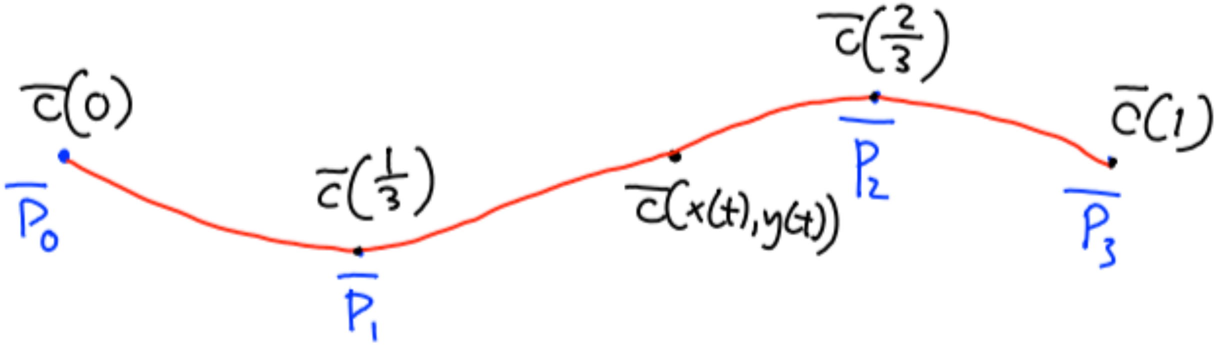
<u>given</u> $\overline{P_1}, \overline{P_2}, \overline{P_3}, \overline{P_4}$

<u>compute</u> a_i, b_i

$$\overset{\mathbf{C}}{\underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\text{known}}} = \overset{\mathbf{A}}{\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1/3 & (1/3)^2 & (1/3)^3 \\ 1 & 2/3 & (2/3)^2 & (2/3)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{known}}} \overset{\mathbf{X}}{\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \\ a_4 & b_4 \end{bmatrix}}_{\text{unknown}}} \Longrightarrow$$

solve system in terms of unknown matrix

$$\mathbf{X} = \mathbf{A}^{-1}\mathbf{C}$$



What if < 4 Control Points?

$$\left.\begin{array}{l} x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ y(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^2 \end{array}\right\} \quad \begin{array}{l} \underline{\text{given}} \; \bar{P_1}, \bar{P_2}, \bar{P_3}, \bar{P_4} \\ \underline{\text{compute}} \; a_i, b_i \end{array}$$

$$\begin{array}{c} \uparrow \\ \text{degree} \\ +1 \\ \downarrow \end{array} \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \\ a_4 & b_4 \end{bmatrix} = \underset{\longleftarrow \; \#\text{control points.} \; \longrightarrow}{\begin{bmatrix} \text{more unknowns} \\ \text{than Eqs} \Rightarrow \\ \text{cannot compute} \\ \text{inverse} \end{bmatrix}^{-1}} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}$$

$$\begin{bmatrix} x_1 & y_1 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{3} & \left(\frac{1}{3}\right)^2 & \left(\frac{1}{3}\right)^3 \end{bmatrix} \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix}$$

What if > 4 Control Points?

$$\left.\begin{aligned} x(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ y(t) &= b_0 + b_1 t + b_2 t^2 + b_3 t^2 \end{aligned}\right\} \quad \begin{aligned} &\underline{\text{given}}\ \overline{p_1}, \overline{p_2}, \overline{p_3}, \overline{p_4} \\ &\underline{\text{compute}}\ a_i, b_i \end{aligned}$$

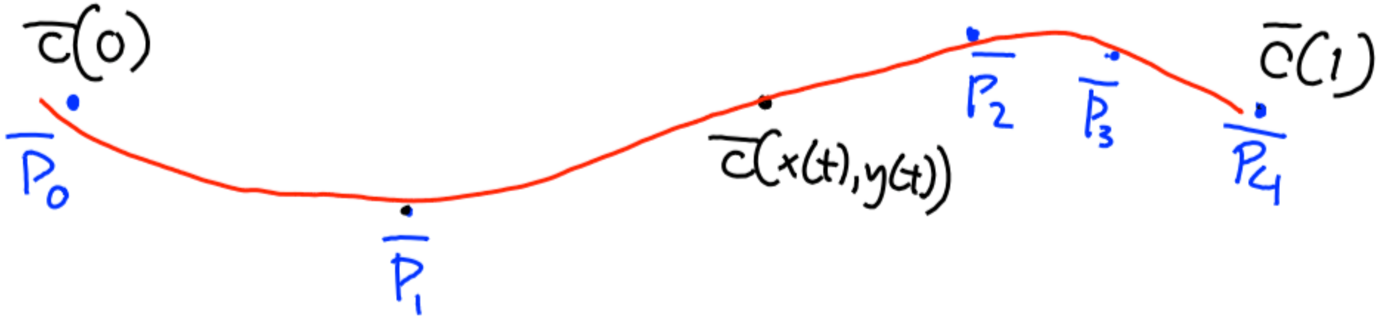
$$\begin{array}{c} \uparrow \\ \text{degree} \\ +1 \\ \downarrow \end{array} \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \\ a_4 & b_4 \end{bmatrix} = \underset{\longleftarrow\ \#\text{control points.}\ \longrightarrow}{\begin{bmatrix} \text{over-determined} \\ \text{linear system} \\ \Rightarrow \\ \text{poly cannot pass} \\ \text{through all pts} \end{bmatrix}^{-1}} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}$$

Degree-N Poly Interpolation

- To interpolate N points perfectly with a single polynomial, we need a polynomial of degree N-1

Major drawback: it is a global interpolation scheme

i.e. moving one control point changes the interpolation of all points, often in unexpected, unintuitive and undesirable ways

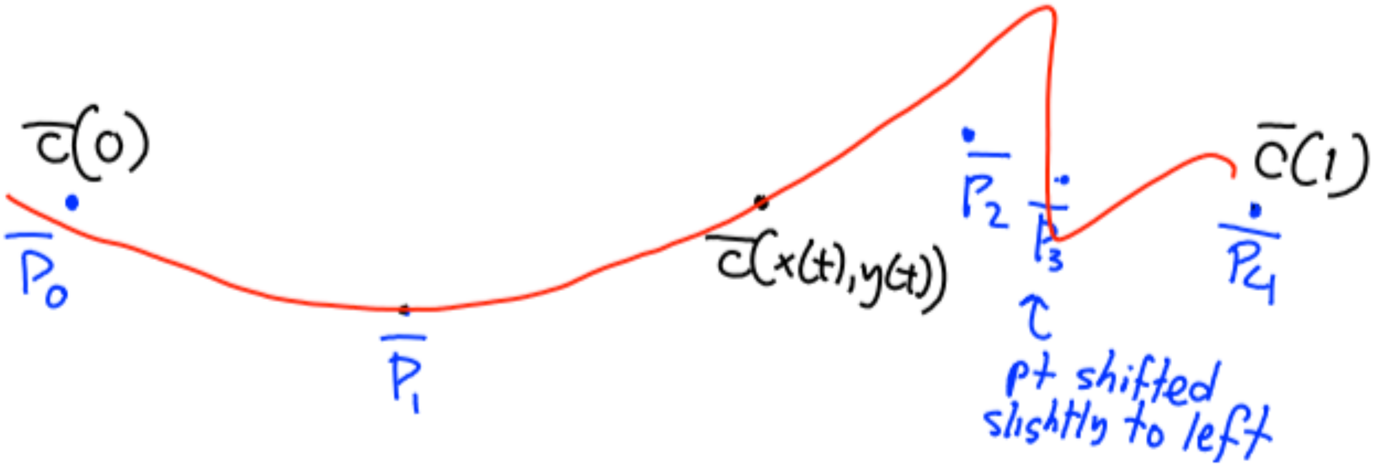


Degree-N Poly Interpolation

- To interpolate N points perfectly with a single polynomial, we need a polynomial of degree N-1

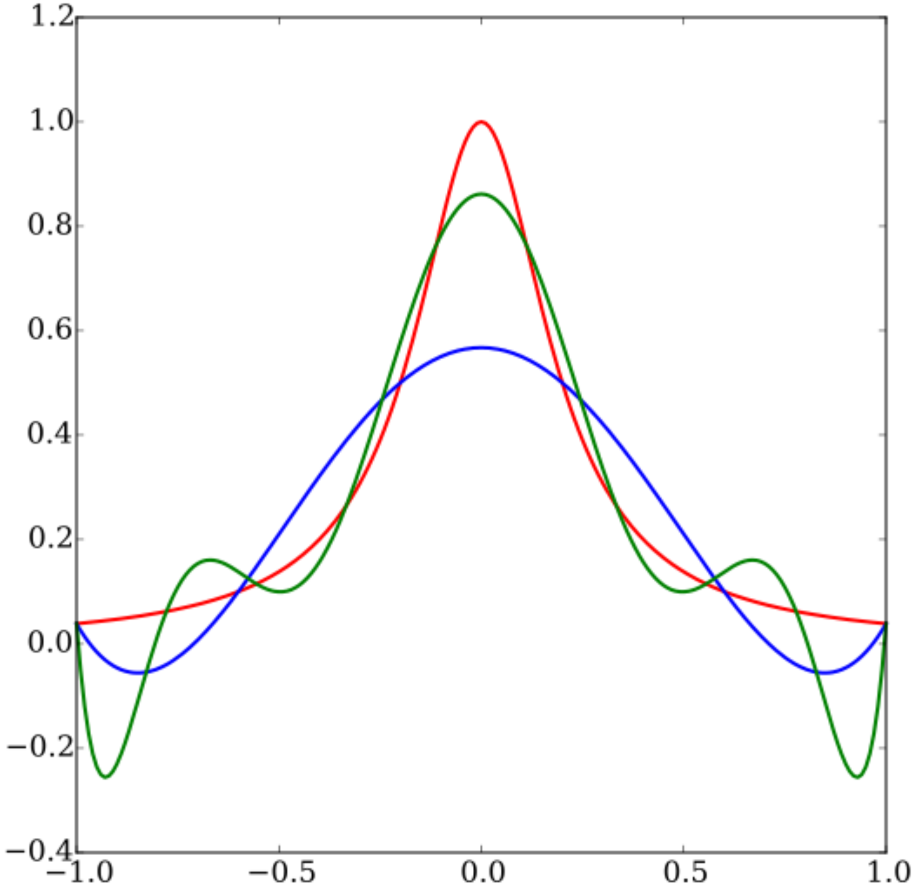
Major drawback: it is a global interpolation scheme

i.e. moving one control point changes the interpolation of all points, often in unexpected, unintuitive and undesirable ways



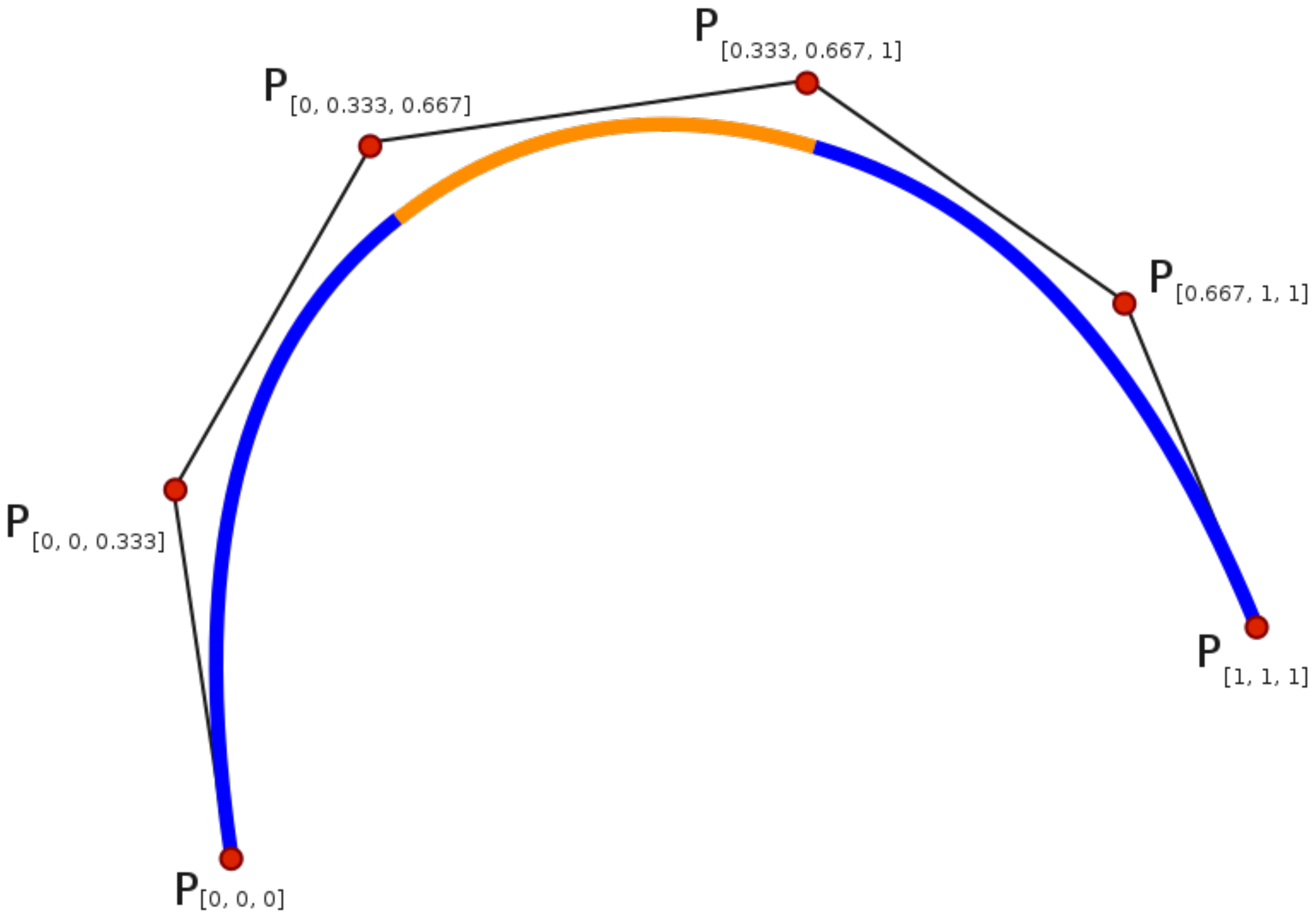
Runge’s Phenomenon

The higher-order the polynomial, the more oscillation you get at the boundaries when using equidistant control points



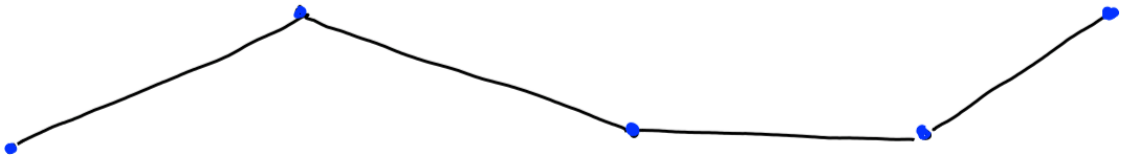
Instead we use "Splines"

Curve is defined by piecewise polynomials



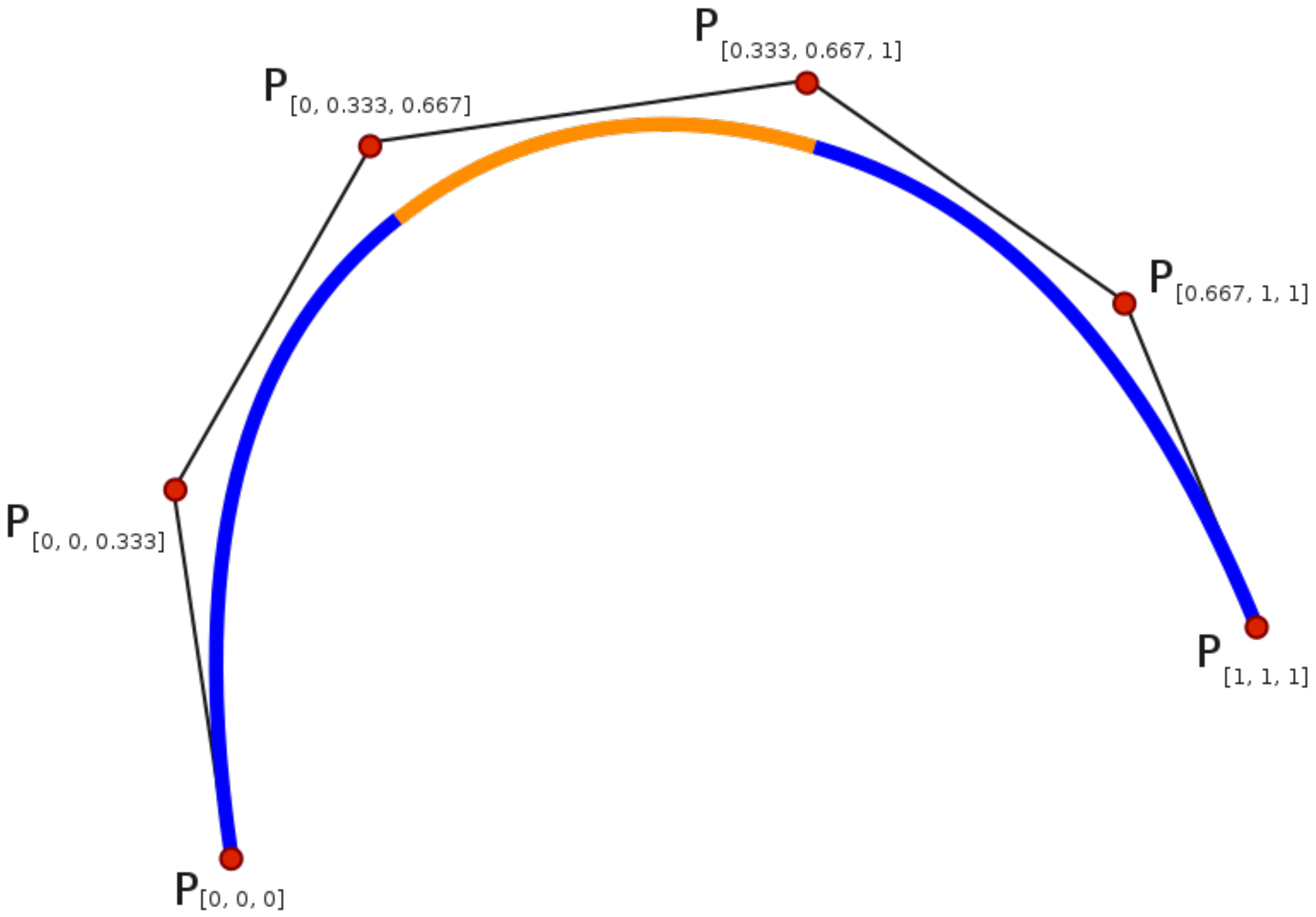
Example: Linear Interpolation

- The simplest possible interpolation technique
- Create a piecewise linear curve that connects the control points



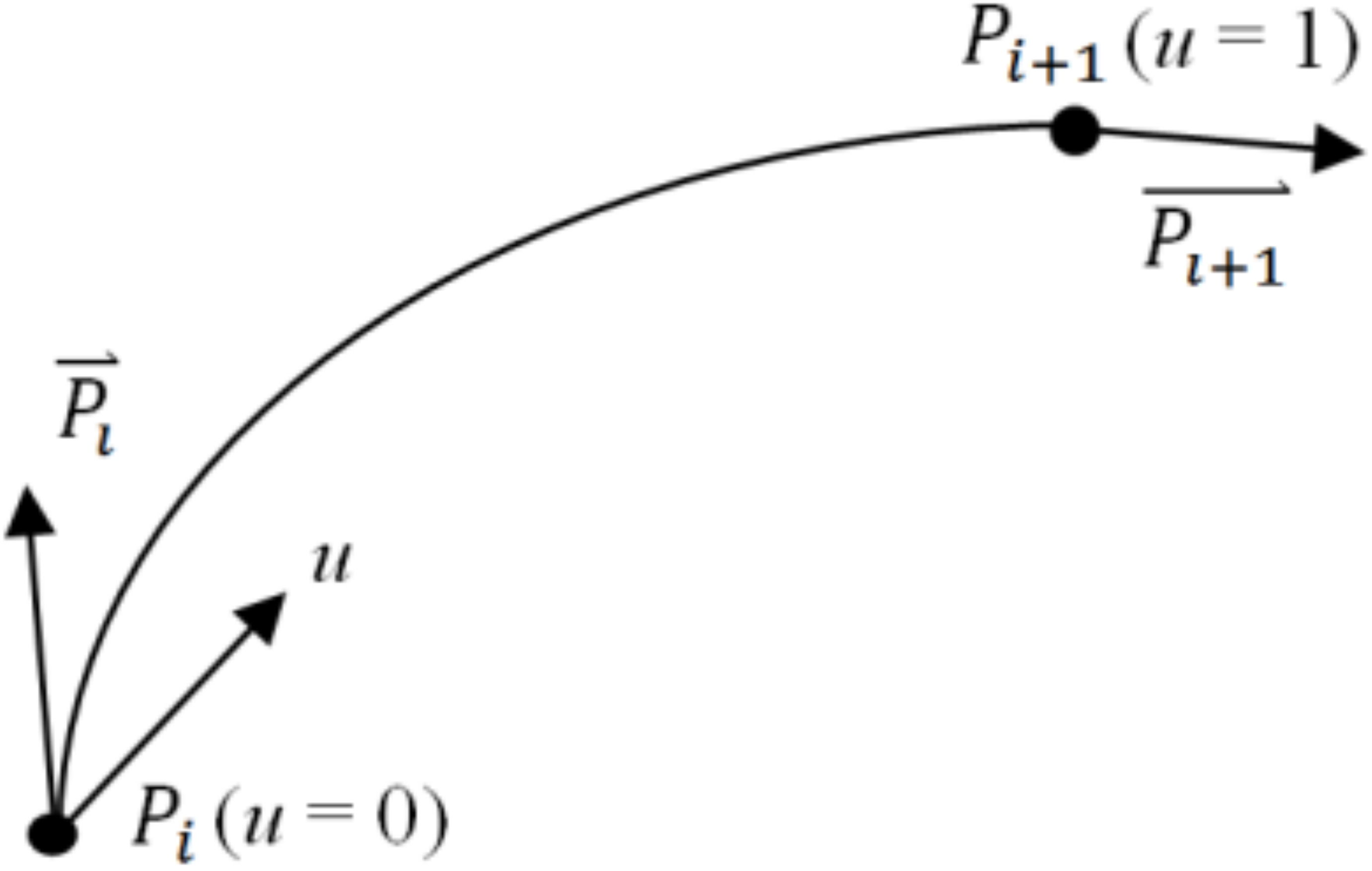
Instead we use "Splines"

Curve is defined by piecewise polynomials



Hermite Splines

- Cubic polynomials specified by end point positons and end point tangents (4 pieces of information)



Evaluating Derivatives

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

$$\frac{dx}{dt}(t) = a_1 + 2a_2 t + 3a_3 t^2$$

$$\boxed{\begin{bmatrix} \frac{dx}{dt}(t) & \frac{dy}{dt}(t) \end{bmatrix} = \begin{bmatrix} 1 & 2t & 3t^2 \end{bmatrix} \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix}}$$

Designing Polynomial Curves from constraints

p(t) = TA , where T is powers of t. for a cubic T=[t^3 t^2 t^1 1].

Written with geometric constraints p(t) = TMG, where M is the **Basis matrix** of a design curve and G the specific design constraints.

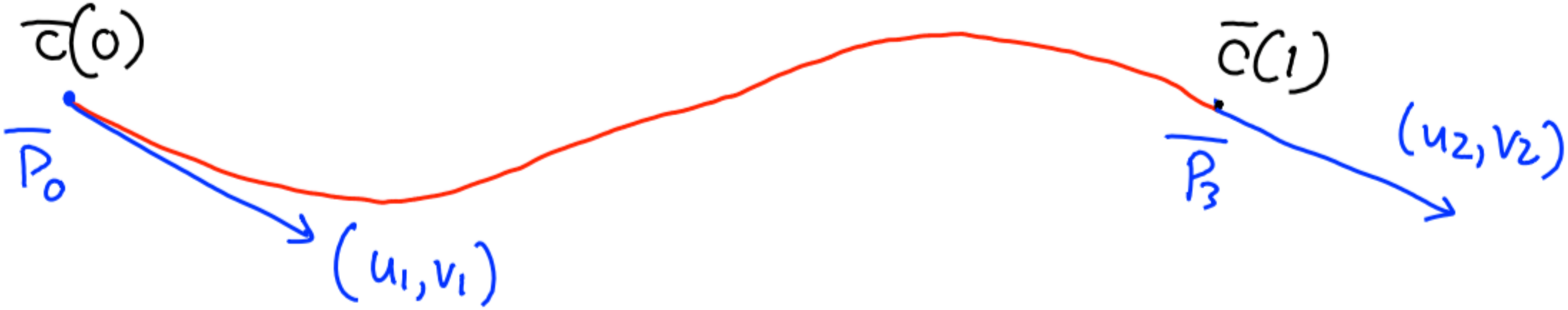
An example of constraints for a cubic Hermite for eg. are end points and end tangents. i.e. P_1,R_1 at t=0 and P_4,R_4 at t=1. Plugging these constraints into p(t) = TA we get.

$$
\begin{array}{llll}
 & & B & \\
p(0) = P_1 & = & [\ 0\ 0\ 0\ 1\ ]\ A_h & \\
p(1) = P_4 & = & [\ 1\ 1\ 1\ 1\ ]\ A_h & \\
p'(0) = R_1 & = & [\ 0\ 0\ 1\ 0\ ]\ A_h & \Rightarrow \quad G=BA,\ A=MG \Rightarrow M=B^{-1} \\
p'(1) = R_4 & = & [\ 3\ 2\ 1\ 0\ ]\ A_h &
\end{array}
$$

Bézier Curves

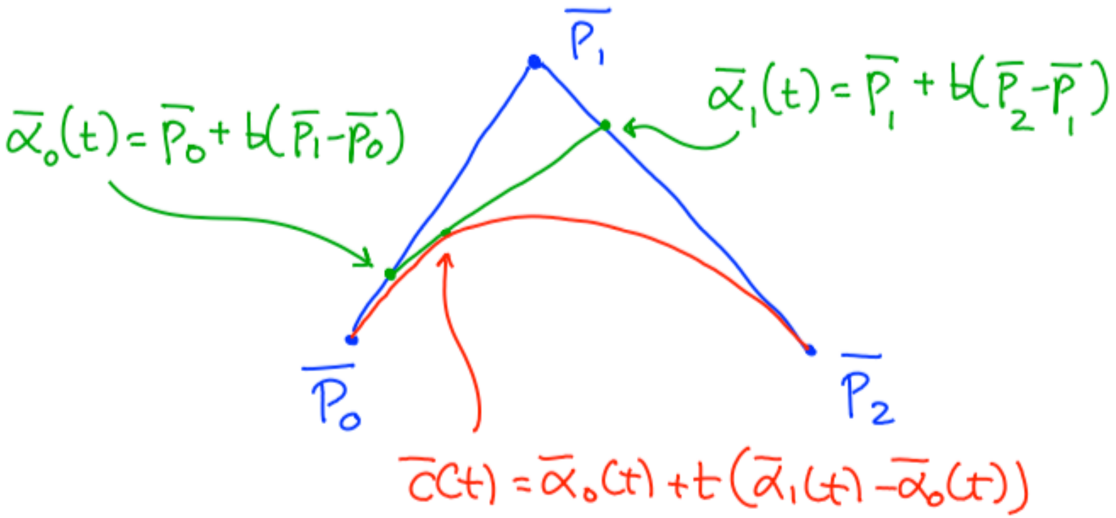
Properties:

- Polynomial curves defined via endpoints and derivative constraints
- Derivative constraints defined implicitly through extra control points (that are not interpolated)
- They are approximating curves, not interpolating curves



Bézier Curves: Main Idea

Polynomial and its derivatives expressed as a cascade of linear interpolations


$\overline{P}_1$
$\overline{\alpha}_1(t) = \overline{P}_1 + t(\overline{P}_2 - \overline{P}_1)$
$\overline{\alpha}_0(t) = \overline{P}_0 + t(\overline{P}_1 - \overline{P}_0)$
$\overline{P}_0$
$\overline{P}_2$
$\overline{c}(t) = \overline{\alpha}_0(t) + t(\overline{\alpha}_1(t) - \overline{\alpha}_0(t))$

algorithm:

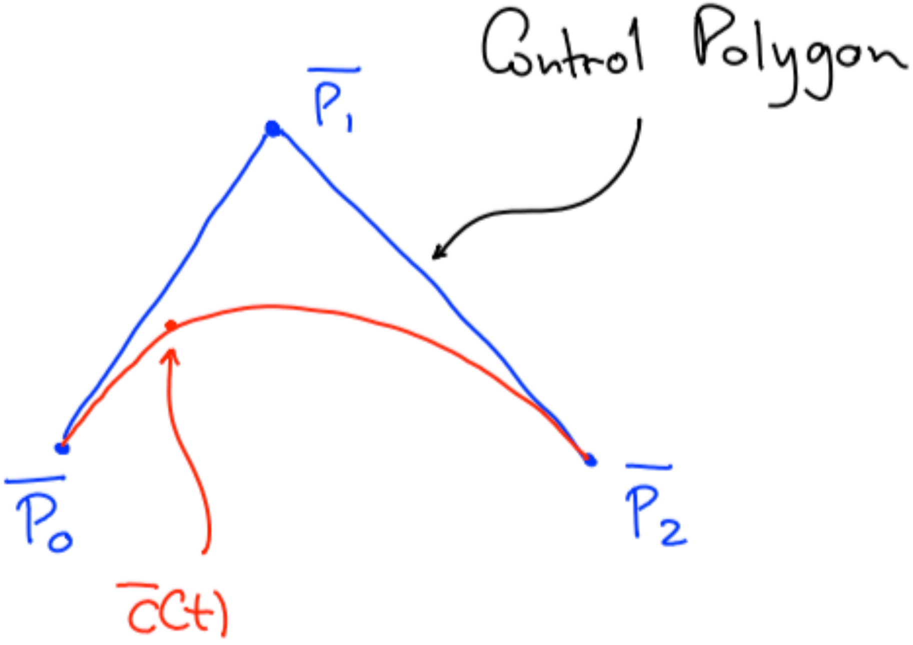
given $\overline{P}_0, \overline{P}_1, \overline{P}_2$ and t

1. linearly interpolate $\overline{P}_0, \overline{P}_1$ to get $\overline{\alpha}_0(t)$
2. lineary interpolate $\overline{P}_1, \overline{P}_2$ to get $\overline{\alpha}_1(t)$
3. linearly interpolate $\overline{\alpha}_0(t), \overline{\alpha}_1(t)$ to get $\overline{c}(t)$

Bézier Curves: Control Polygon

A Bézier curve is completely determined by its control polygon

We manipulate the curve by manipulating its polygon



```
algorithm:
  given $\bar{P}_0, \bar{P}_1, \bar{P}_2$ and $t$
    1. linearly interpolate
       $\bar{P}_0, \bar{P}_1$ to get $\bar{\alpha}_0(t)$
    2. lineary interpolate
       $\bar{P}_1, \bar{P}_2$ to get $\bar{\alpha}_1(t)$
    3. linearly interpolate
       $\bar{\alpha}_0(t), \bar{\alpha}_1(t)$ to
       get $\bar{c}(t)$
```

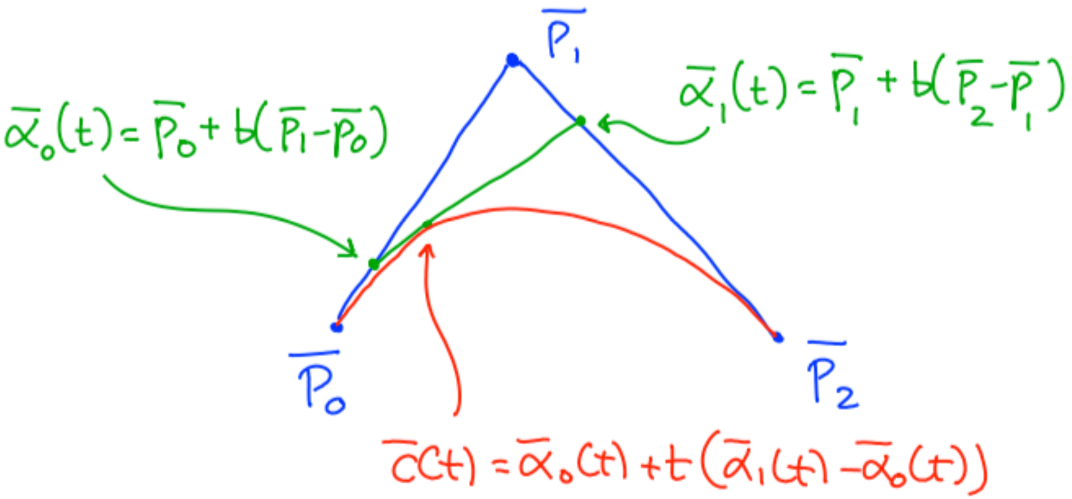
Bézier Curve as a Polynomial

Computing the polynomial

$$c(t) = [P_0 + t(\bar{P}_1 - \bar{P}_0)] + t[\bar{P}_1 + t(\bar{P}_2 - \bar{P}_1) - \bar{P}_0 - t(\bar{P}_1 - \bar{P}_0)]$$

$$= \bar{P}_0(1 - t - t + t^2) + \bar{P}_1(t + t - t^2 - t^2) + \bar{P}_2 t^2$$

$$= \bar{P}_0(1-t)^2 + 2\bar{P}_1 t(1-t) + \bar{P}_2 t^2$$


$\bar{P}_1$
$\bar{\alpha}_1(t) = \bar{P}_1 + t(\bar{P}_2 - \bar{P}_1)$
$\bar{\alpha}_0(t) = \bar{P}_0 + t(\bar{P}_1 - \bar{P}_0)$
$\bar{P}_0$
$\bar{P}_2$
$\bar{c}(t) = \bar{\alpha}_0(t) + t(\bar{\alpha}_1(t) - \bar{\alpha}_0(t))$

algorithm:

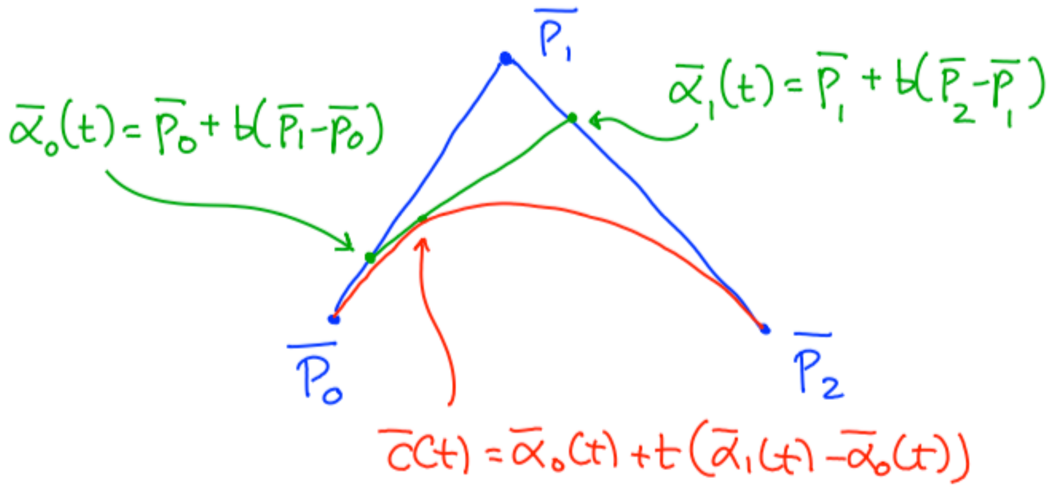
given $\bar{P}_0, \bar{P}_1, \bar{P}_2$ and t

1. linearly interpolate $\bar{P}_0, \bar{P}_1$ to get $\bar{\alpha}_0(t)$
2. lineary interpolate $\bar{P}_1, \bar{P}_2$ to get $\bar{\alpha}_1(t)$
3. linearly interpolate $\bar{\alpha}_0(t), \bar{\alpha}_1(t)$ to get $\bar{c}(t)$

Derivatives of the Bézier Curve

Computing the polynomial's derivatives:

$$\frac{d}{dt}C(t) = -2(1-t)\,P_0 + 2P_1(1-2t) + P_2\,2t \quad \begin{aligned} &\approx 2(P_1-P_0) \text{ at } t=0 \\ &\approx 2(P_2-P_1) \text{ at } t=1 \end{aligned}$$



$\bar{P}_1$
$\bar{\alpha}_1(t) = \bar{P}_1 + t(\bar{P}_2 - \bar{P}_1)$
$\bar{\alpha}_0(t) = \bar{P}_0 + t(\bar{P}_1 - \bar{P}_0)$
$\bar{P}_0$
$\bar{P}_2$
$\bar{c}(t) = \bar{\alpha}_0(t) + t(\bar{\alpha}_1(t) - \bar{\alpha}_0(t))$

algorithm:

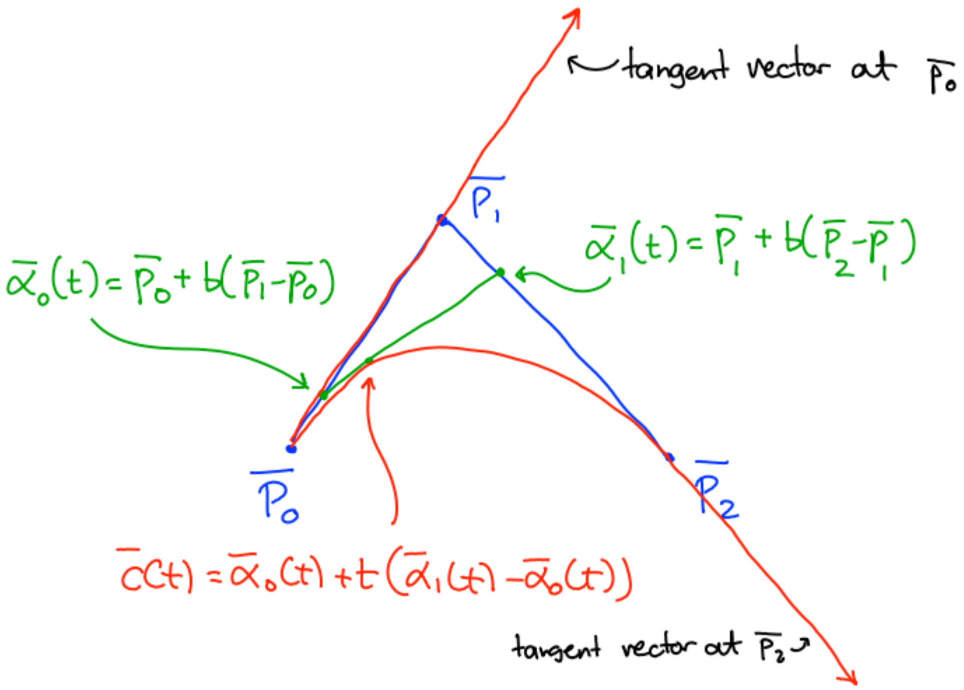
given $\bar{P}_0, \bar{P}_1, \bar{P}_2$ and t

1. linearly interpolate $\bar{P}_0, \bar{P}_1$ to get $\bar{\alpha}_0(t)$
2. linearly interpolate $\bar{P}_1, \bar{P}_2$ to get $\bar{\alpha}_1(t)$
3. linearly interpolate $\bar{\alpha}_0(t), \bar{\alpha}_1(t)$ to get $\bar{c}(t)$

Endpoints and Tangent Constraints

Computing the polynomial's derivatives:

$$\frac{d}{dt}c(t) = -2(1-t)P_0 + 2P_1(1-2t) + P_2\,2t \begin{cases} = 2(p_1-p_0) & \text{at } t=0 \\ = 2(P_2-P_1) & \text{at } t=1 \end{cases}$$



← tangent vector at $\bar{P}_0$
$\bar{P}_1$
$\bar{\alpha}_1(t) = \bar{P}_1 + t(\bar{P}_2 - \bar{P}_1)$
$\bar{\alpha}_0(t) = \bar{P}_0 + t(\bar{P}_1 - \bar{P}_0)$
$\bar{P}_0$
$\bar{P}_2$
$\bar{c}(t) = \bar{\alpha}_0(t) + t(\bar{\alpha}_1(t) - \bar{\alpha}_0(t))$
tangent vector at $\bar{P}_2$ ↗

General Behaviour

- 1st and 3rd control points define the endpoints.
- 2nd control point defines the tangent vector at the endpoints.

Generalization to N+1 points

Expression in compact form:

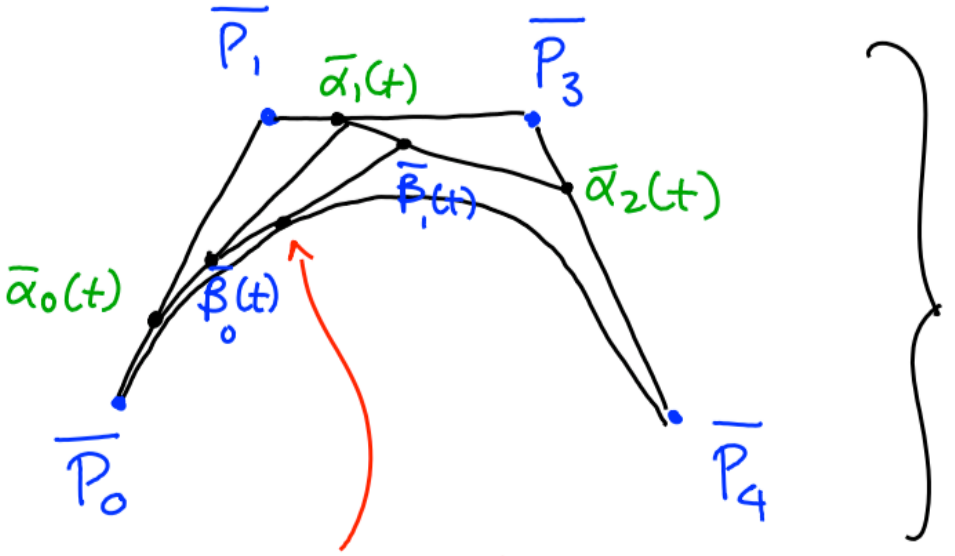
$$\underset{\substack{\uparrow\\\text{curve}}}{\bar{c}(t)} = \sum_{i=0}^{N} \underset{\substack{\uparrow\\\text{control pt}}}{\bar{P}_i} B_i^N(t)$$

called the Bernstein Polynomials of degree N

$$B_i^N(t) = \binom{N}{i}(1-t)^{N-i}\,t^i$$

$$= \frac{N!}{(N-i)!\,i!}(1-t)^{N-i}\,t^i$$

Curve defined by N linear interpolation cascades (De Casteljau's algorithm):



$$\bar{c}(t) = \bar{\beta}_0(t) + t\left(\bar{\beta}_1(t) - \bar{\beta}_0(t)\right)$$

Example for 4 control points and 3 cascades

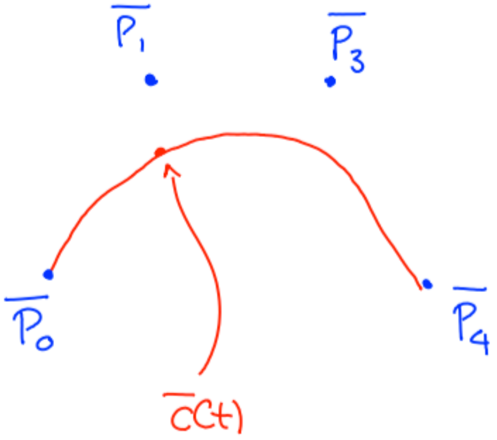
Bézier and Control Points

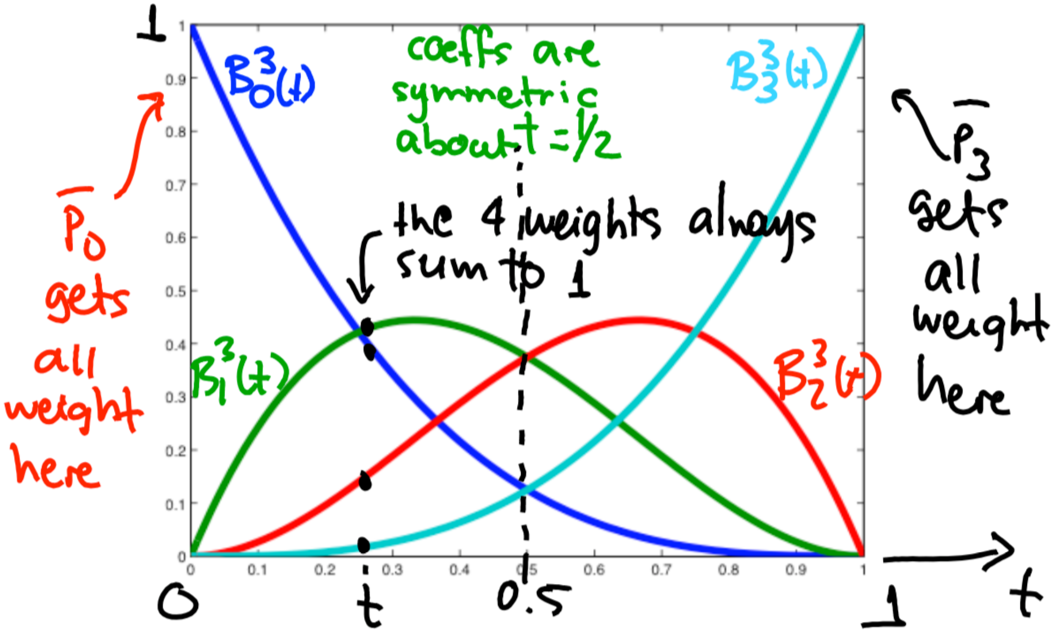
Expression in compact form:

$$\bar{c}(t) = \sum_{i=0}^{N} \bar{P}_i B_i^N(t)$$

curve

control pt





with $\sum_{i=0}^{N} B_i^N(t) = 1$ for all t

Bézier Curves: Useful Properties

Expression in compact form:

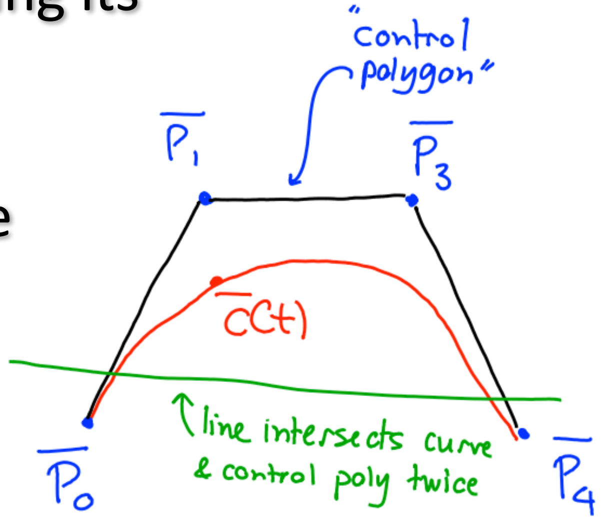
$$\bar{c}(t) = \sum_{i=0}^{N} \bar{p}_i B_i^N(t)$$

called the Bernstein Polynomials of degree N

$$B_i^N(t) = \binom{N}{i}(1-t)^{N-i} t^i$$

$$= \frac{N!}{(N-i)!\,i!}(1-t)^{N-i} t^i$$

1. Affine Invariance
   - Transforming a Bézier curve by an affine transform T is equivalent to transforming its control points by T
2. Diminishing Variation
   - No line will intersect the curve at more points than the control polygon
     - curve cannot exhibit “excessive fluctuations”
3. Linear Precision
   - If control poly approximates a line, so will the curve



Bézier Curves: Useful Properties

Expression in compact form:

$$\bar{c}(t) = \sum_{i=0}^{N} \bar{P}_i B_i^N(t)$$

called the Bernstein Polynomials of degree N

$$B_i^N(t) = \binom{N}{i} (1-t)^{N-i} t^i$$

$$= \frac{N!}{(N-i)!\, i!} (1-t)^{N-i} t^i$$

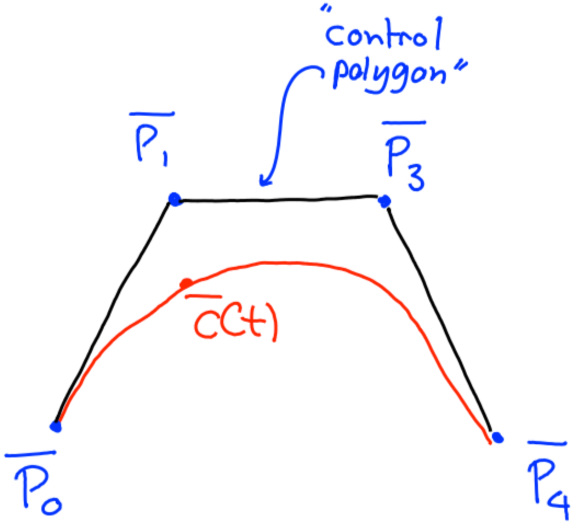
4. Tangents at endpoints are along the 1st and last edges of control polygon:

$$\frac{d}{dt}\bar{c}(t) = \sum_{i=1}^{N} \bar{P}_i \frac{d}{dt} B_i^N(t)$$

$$\overset{\text{w/ some work}}{=} N \sum_{i=0}^{N-1} (\bar{P}_{i+1} - \bar{P}_i) B_i^{N-1}(t)$$

$\parallel$ $N(\bar{P}_1 - \bar{P}_0)$ for $t=0$

$\parallel$ $N(\bar{P}_N - \bar{P}_{N-1})$ for $t=1$



Bézier Curves: Pros and Cons

Advantages:

- Intuitive control for N ≤ 3
- Derivatives easy to compute
- Nice properties (affine invariance, diminishing variation)

Disadvantages:

- Scheme is still global (curve is function of all control points)

Reminders

Bezier Basis Matrix

A cubic Bezier can be defined with four points where:
P_1,R_1 at t=0 and P_4,R_4 at t=1 for a Hermite.
$R_1 = 3(P_2-P_1)$ and $R_4 = 3(P_4-P_3)$.

We can thus compute the Bezier Basis Matrix by finding the matrix that transforms $[P_1\ P_2\ P_3\ P_4]^T$ into $[P_1\ P_4\ R_1\ R_4]^T$ i.e.

B_H =[1 0 0 0]
[0 0 0 1]
[-3 3 0 0]
[0 0 -3 3]

$M_{bezier}=M_{hermite} *$ B_H

Bezier Basis Functions

$$\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The columns of the Basis Matrix form Basis Functions such that:
$p(t)= f_1(t)P_1 + f_2(t)P_2 + f_3(t)P_3 + f_4(t)P_{4.}$

From the matrix:

$$f_i(t) = \binom{n}{i} \ast (1-t)^{(n-i)} \ast t^i$$

These are also called Bernstein polynomials.

Basis Functions

Basis functions can be thought of as interpolating functions.
Note: actual interpolation of any point only happens if its Basis function is 1 and all others are zero at some t.

Often Basis functions for design curves sum to 1 for all t.
This gives the curve some nice properties like affine invariance and the convex hull property when the function are additionally non-negative.