Lecture 10:

More Curves

Last Time...

- Definitions
- Kinds of Curves
- Parametric Curves
- Continuity
- Polynomial Forms

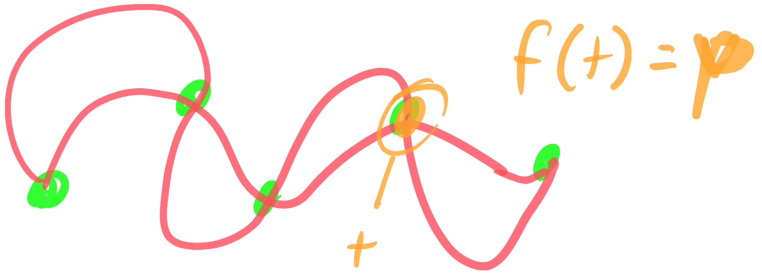
Today

- Polynomial Curves!
- Interpolation
- Basis Function Forms
- Cubics
- Hermite Interpolation
- Cardinal Interpolation
  Catmull-Rom Splines
- Beziers

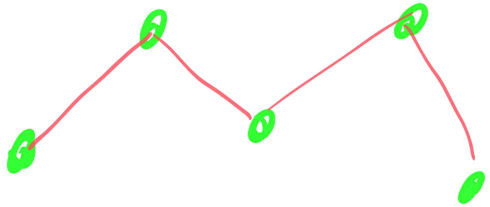
Interpolation

- Specify values (with corresponding t)

How to do this?



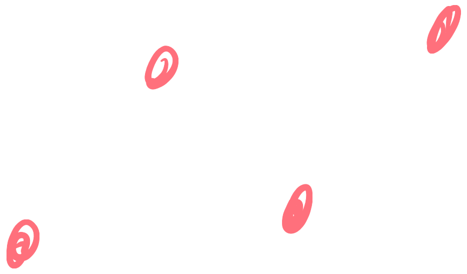
- line segments - C(0)
- polynomial fitting - (in book, demo) - C(∞)
- piecewise polynomials
  - Hermite Cubics C(1) (special tangents)
  - Cardinal Cubics C(1)
  - C(2) interpolation is harder - won't discuss today



n points
n-1 degree polynomial

Linear Intepolation

- Piecewise polynomial
- Local control
- Predict in-between
- only C[0]



Polynomial Fit

- Single polynomial
- No local control
- Hard to predict
- $C(\infty)$

A Simple Polynomial (a line)

$$\mathbf{f}(u) = \mathbf{a_0} + \mathbf{a_1} u$$

Note: $\mathbf{a}_0$ and $\mathbf{a}_1$ are in 2D

Specify a Line

Make a line between $\mathbf{p}_0$ and $\mathbf{p}_1$

$$\mathbf{f}(0) = \mathbf{p}_0$$
$$\mathbf{f}(1) = \mathbf{p}_1$$

We can figure out the coefficients...

$\mathbf{f}(0) = \mathbf{a}_0 + \mathbf{a}_1 0$ (since u=0) so $\mathbf{a}_0 = \mathbf{p}_0$

$\mathbf{f}(1) = \mathbf{a}_0 + \mathbf{a}_1 1$ (since u=1) so $\mathbf{p}_1 = \mathbf{a}_0 + \mathbf{a}_1$ or $\mathbf{a}_1 = \mathbf{p}_1 - \mathbf{a}_0$

A convenient form to write it in...

Who needs the coefficients? (do a little algrebra)

$$\mathbf{f}(u) = (1-u)\mathbf{p_0} + u\mathbf{p_1}$$

Note that we've written the function in terms of "control points"

We could write this as a function for each point...

$$\mathbf{f}(u) = b_0(u)\mathbf{p_0} + b_1(u)\mathbf{p_1}$$

BASIS

where...

$$b_0(u) = (1-u) \qquad b_1(u) = u$$

Basis Functions

Write functions in terms of "control points"

Write a **basis function** for each control point

$$\mathbf{f}(u) = b_0(u)\mathbf{p_0} + b_1(u)\mathbf{p_1} + b_2(u)\mathbf{p_2} \cdots$$

Polynomials can be written this way

Some things to note...

- the functions are scalar functions, and only depend on u
- there is a separate function for each point
- if we know how to compute the functions, we can plug in values

Quadratic (2nd degree) Segments

$\mathbf{a_0}$, $\mathbf{a_1}$, and $\mathbf{a_2}$

$$\mathbf{f(u)} = \mathbf{a_0} + \mathbf{a_1} u + \mathbf{a_2} u^2$$

what can we do with this?

specify the beginning

- $\mathbf{f}(0) = \mathbf{a_0}$
- $\mathbf{f}'(0) = \mathbf{a_1}$
- $\mathbf{f}''(0) = 2\,\mathbf{a_2}$

Specify the end?

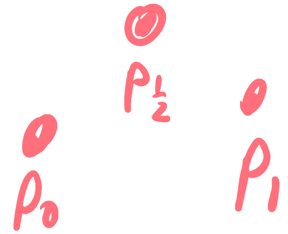
$\mathbf{f(u)} = \mathbf{a_0} + \mathbf{a_1}u + \mathbf{a_2}u^2$

- $\mathbf{f}(1) = \mathbf{a_0} + \mathbf{a_1} + \mathbf{a_2}$
  - if you want to specify where the curve ends, you can compute $\mathbf{a_2}$

We need to specify 3 things... What is convenient?

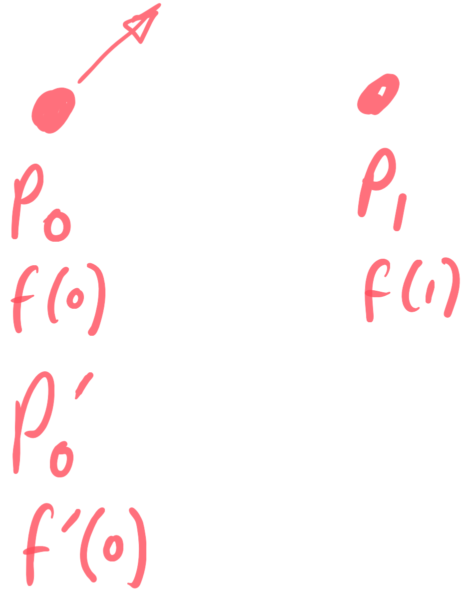
- everything at beginning?
- beginning, end, and... 1 more thing?

Quadratic Interpolation



Note: this is not a common thing, just doing it for pedagogy

- $\mathbf{p}_0$ - position at the beginning
- $\mathbf{p}_1$ - position at the end ($p_1 = a_0 + a_1 + a_2$)



one choice for the third thing...

- $\mathbf{p}_0'$ - derivatrive at the beginning

We can work out the math...

- $a_0 = p_0 \quad ; a_1 = p_0' \quad ; a_2 = p_1 - (p_0 + p_1')$

We can work out the basis functions

$\mathbf{f}(u) = b_0(u)\mathbf{p_0} + b_1(u)\mathbf{p_0'} + b_2(u)\mathbf{p_1}$

- $b_0(u) = (1 - u^2)$
- $b_1(u) = (1 - u)$
- $b_2(u) = u^2$

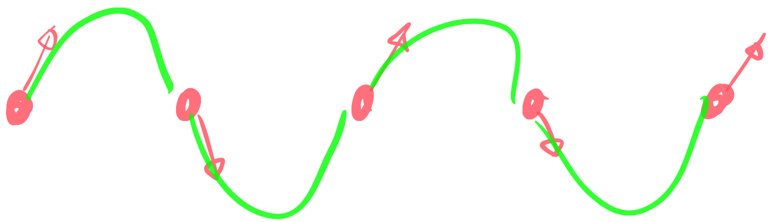
  Don't worry - you don't have to do this

The notation is a little weird... I chose p_0, p_0', p_1, so we have 0,0',1 rather than 0,1,2.

Using this...

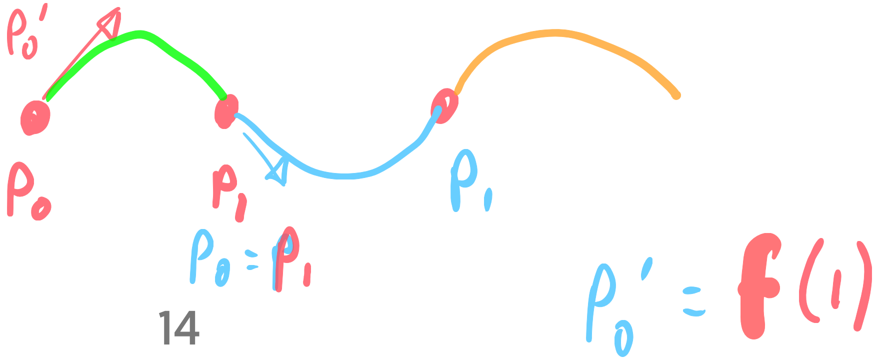
Make a C(0) curve through a bunch of points

- easy make p_0 of segment $n + 1$ same as p_1 of segment n



Make a C(1) curve...

- harder. need to compute the derivative at the end of a segment and use it for the next segment



Cubics

$$\mathbf{f}(\mathbf{u}) = \mathbf{a_0} + \mathbf{a_1}u + \mathbf{a_2}u^2 + \mathbf{a_3}u^3$$

coefficient form is not convenient

Hermite Form

specify position and 1st derivative at ends

$\mathbf{p}_0, \mathbf{p}_1$ as well as $\mathbf{p}'_0, \mathbf{p}'_1$

need to compute $\mathbf{a}_i$ from these

derivation in the book (or old versions of the class)

Hermite Equations

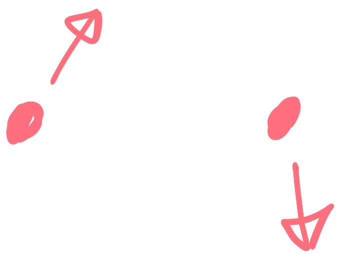
$$
\begin{aligned}
f(u) = p_0\, u^0 &+ \\
p_0'\, u^1 &+ \\
(-3p_0 - 2p_0' + 3p_1 - p_1')\, u^2 &+ \\
(2p_0 + p_0' - 2p_1 - p_1')\, u^3 &
\end{aligned}
$$

so...

$\mathbf{a_0} = \mathbf{p_0}$ and so on...

A more useful form

$$f(t) = (1 - 3u^2 + 2u^3)\, p_0 + \\ (u - 2u^2 + u^3)\, p_0' + \\ (3u^2 - 2u^3)\, p_1 + \\ (-u^2 + u^3)\, p_1'$$



functions of u for each "control point"

$$f(t) = b_0(u)p_0 + b_1(u)p_1 + b_2(u)p_0' + b_3(u)p_1'$$

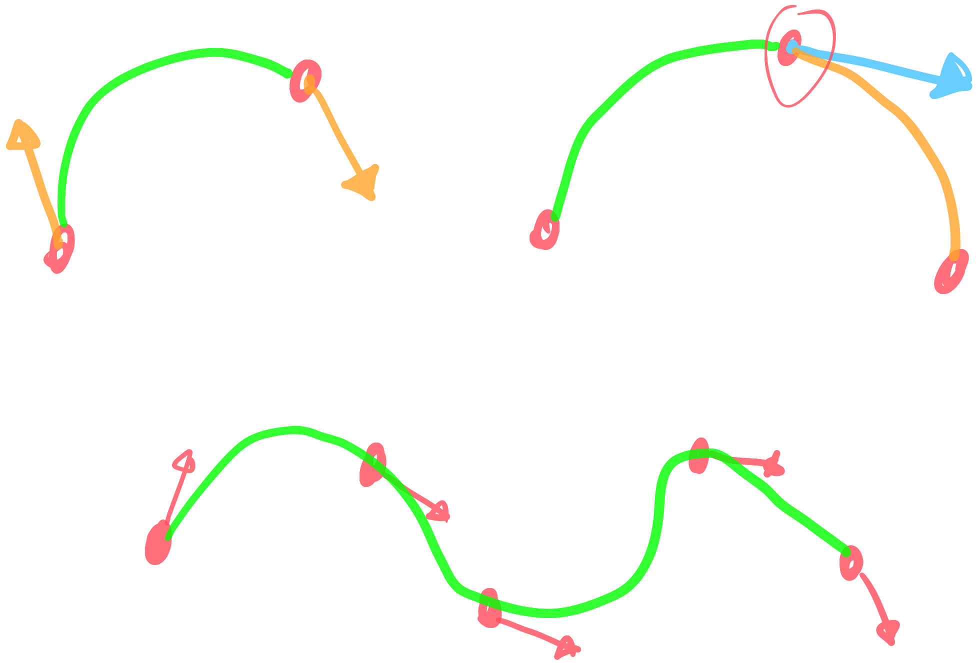
$b_0(u) = 1 - 3u^2 + 2u^3$, etc.

basis functions

Designing with Hermite Curves

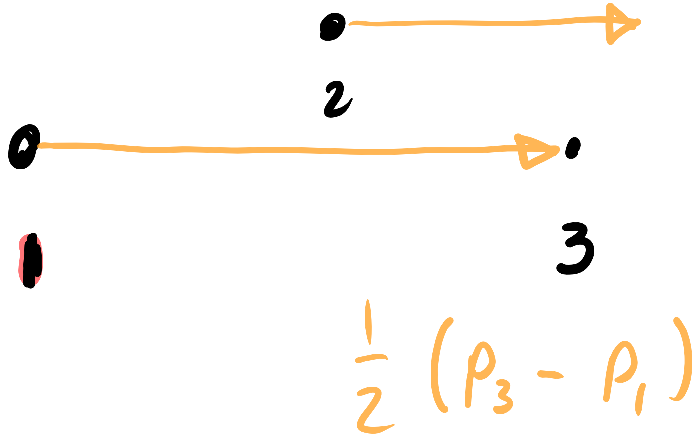
We can make C(1) shapes easily

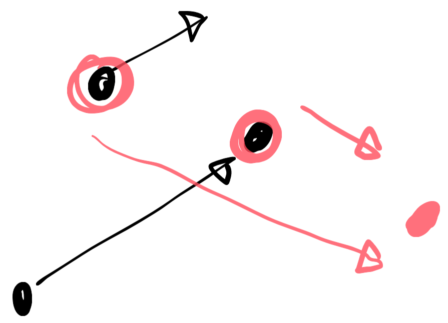
Control "in-between" with derivatives



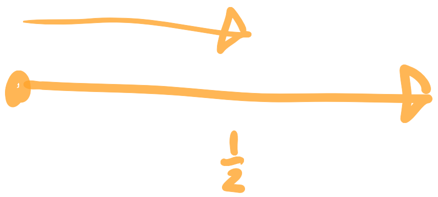
Avoid specifying derivatives?

Compute derivatives based on neighbor points





Cardinal Splines



Catmull-Rom Splines

scaling by 1/2

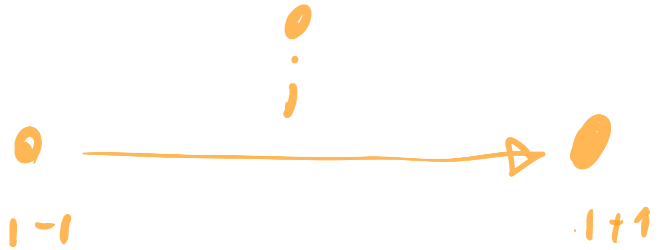
Tension Parameter

$$f_i' = s(f_{i+1} - f_{i-1})$$

$$s = \frac{1-t}{2}$$

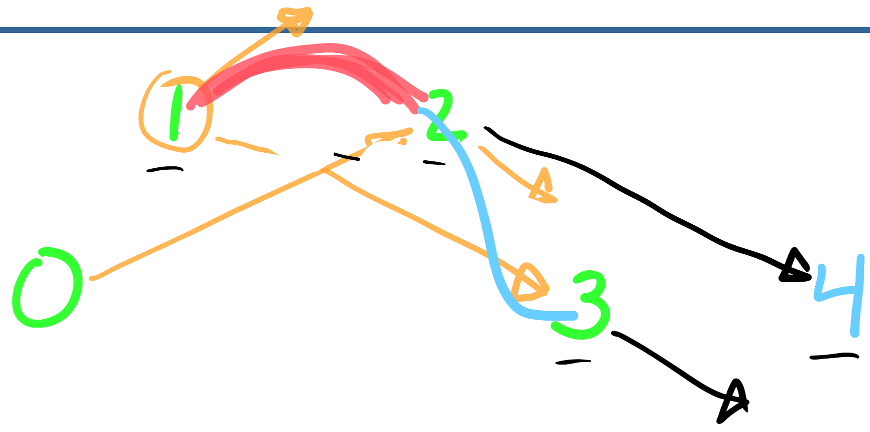
tension

$$t = 0,\ s = \frac{1}{2}$$

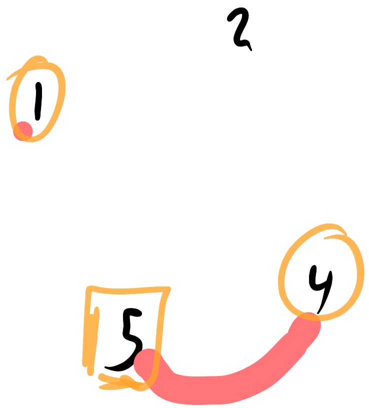


Cardinal Interpolation

- Each segment considers 4 points
- connects 1 to 2
- 0 and 3 used for deriviatives

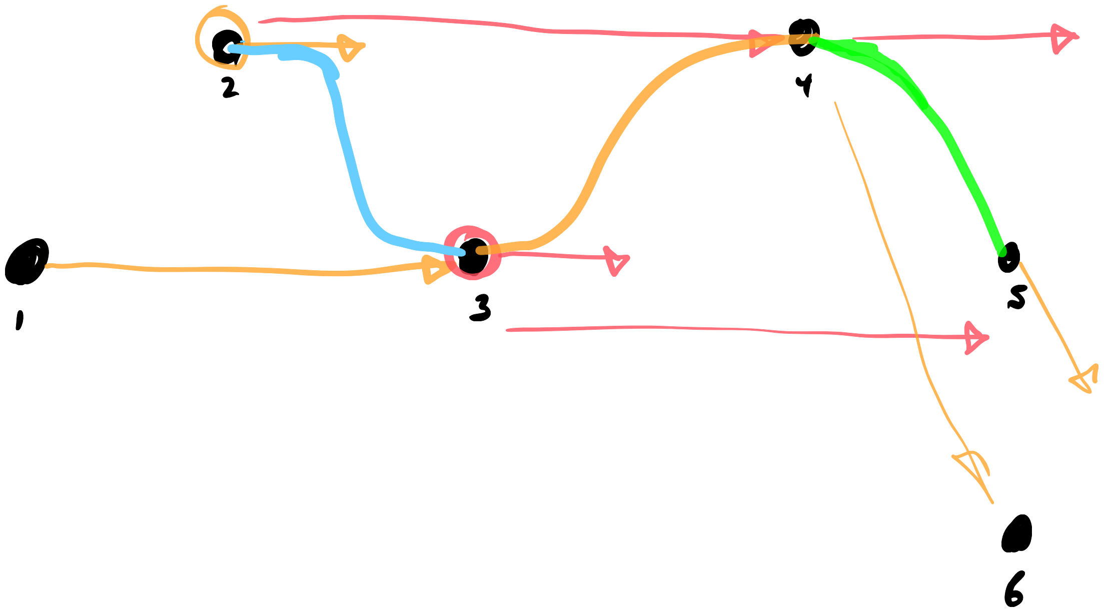


- chain of points - first and last is special
- cycle of points - goes around the loop



- Catmull-Rom is s=1/2 (t=0)

Sketching a Cardinal

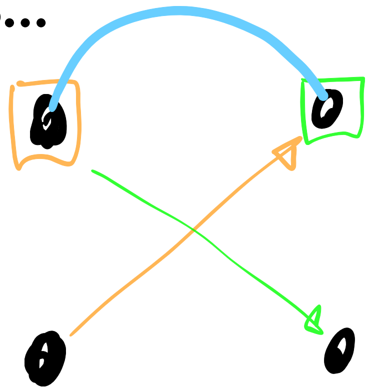


What about not-interpolating?

Why not just interpolate?

- less good control *between* sites

We'll come back to this...



Approximating Curves

How do we use a set of points to control a curve?

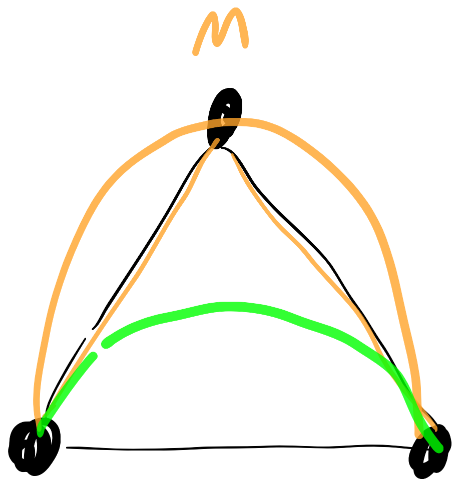
Some points **interpolate**

Other points **influence**

What happens between 2 points?

2 points: connect the dots (line) - or anything else!

Add a third point to influence the shape. What should it do?

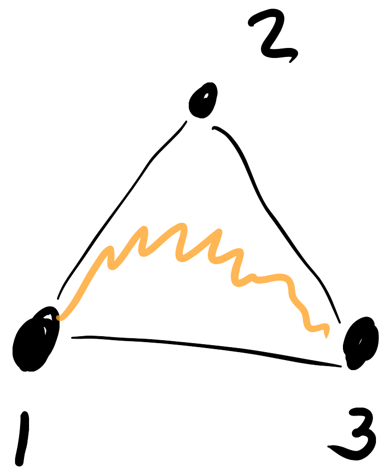


Convenient things for 3 points...

If we are not interpolating the third point...

1. Interpolate the end points
2. Stay inside the triangle
3. Not "wiggle too much"
4. Symmetry (forward/backwards)
5. Locality (only these points)
6. Control tangents (2* vector)

7. Generalize to higher degree (more points)



Bézier Curves

(mispelling warning - no accent is commonly accepted in English)

Some History

Pierre Bézier (Renault):

Bernstein Basis Polynomials

Use polynomials of special form (algrebraic)

Published first

Paul De Castlejau (Citroen):

Geometric Construction

Used simple geometric construction

Bézier figured out its the same thing

Maybe invented first?

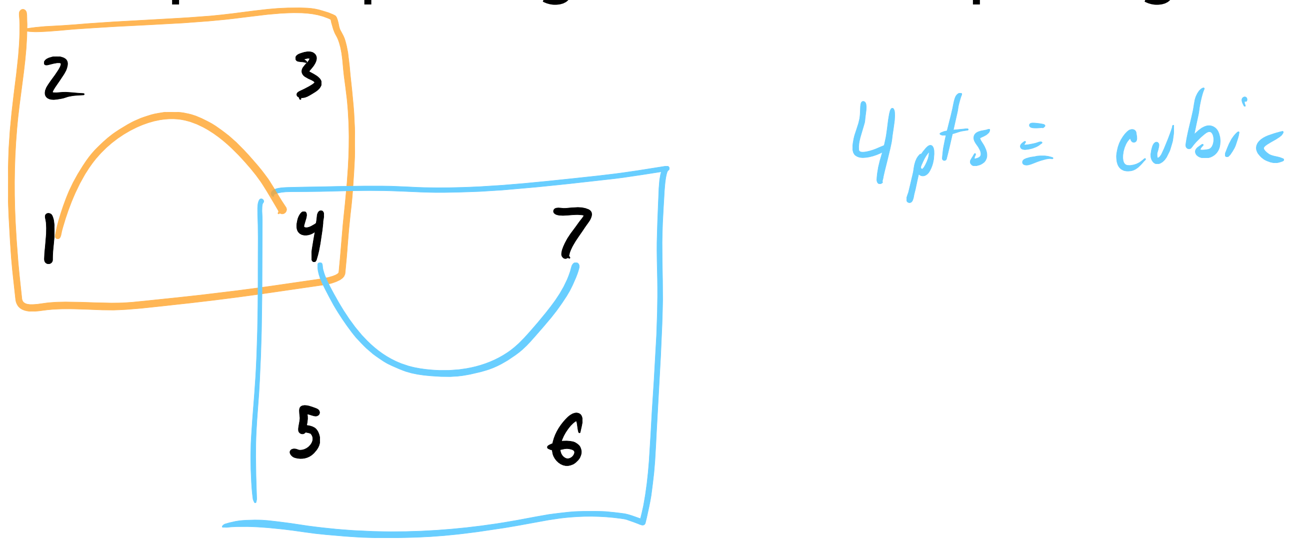
Wasn't allowed to publish

Bézier Curves

Very general - works for any degree

Any number of points per segment (1 more than degree)

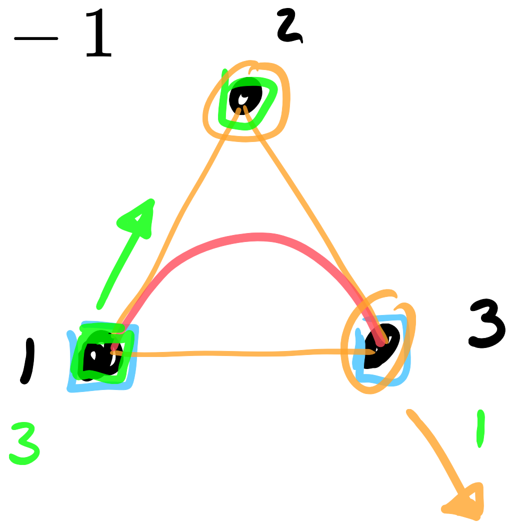
Do not confuse points per segment vs. multiple segments



Quadratic Bézier Curves (3 points)

Three points will give **quadratic** polynomials $d = n - 1$

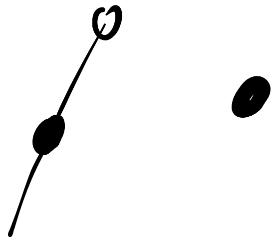
1. Interpolate the end points
2. Stay inside the triangle
3. Not "wiggle too much"
4. Symmetry (forward/backwards)
5. Locality (only these points)
6. Control tangents (first/last two points)



and they generalize to higher degrees

And there's more (we love Béziers)

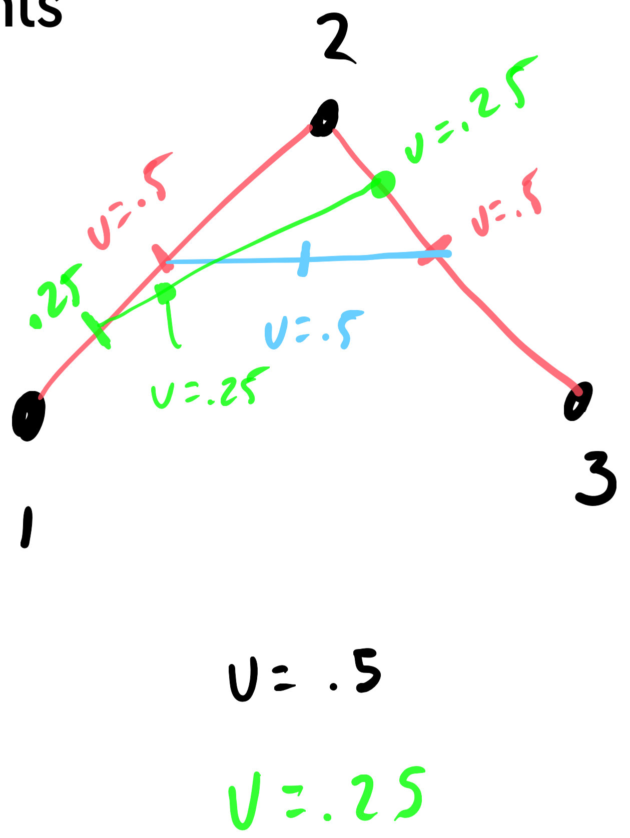
1. Efficient algorithms
2. Common UIs
3. Supported in most APIs
4. Nice mathematical properties
5. Affine Invariance
6. Elegant derivations



The DeCastlejau Construction

Repeated linear interpolation (for the U value)

Try with 3 points

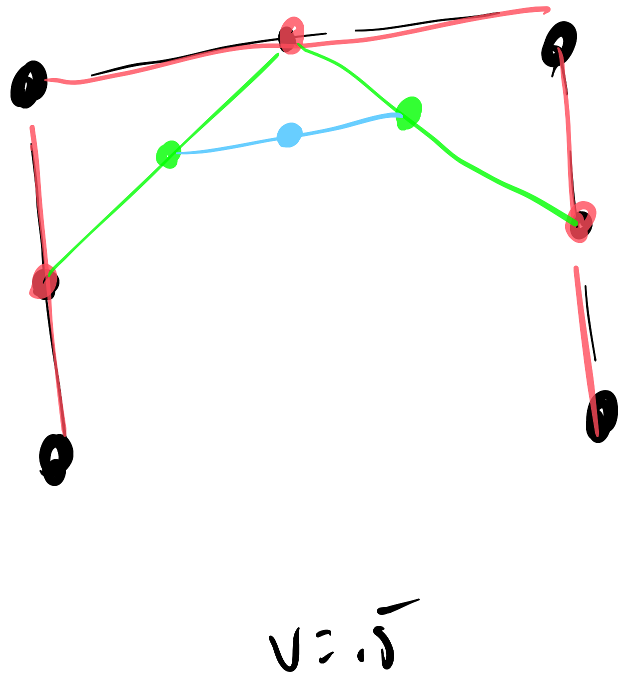


DeCastlejau Construction

For a different u value

The DeCastlejau Construction

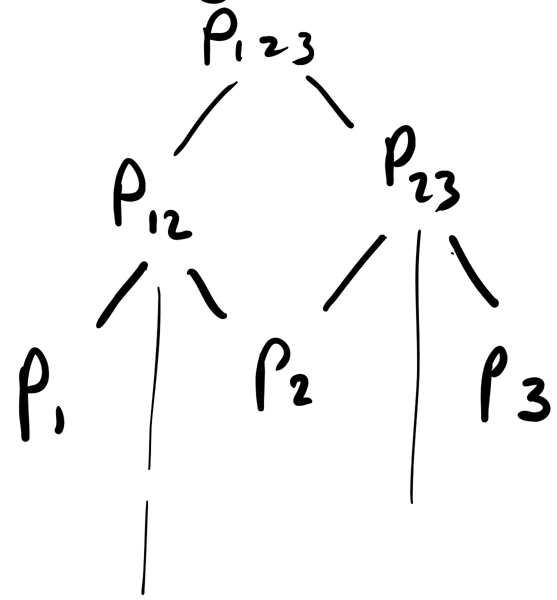
Extends to any number of points



The blending tree

Easy by hand

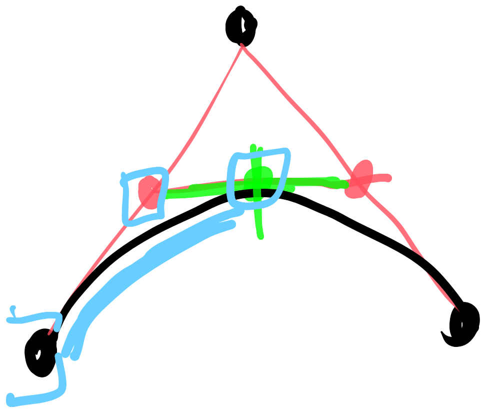
put in U to see algebra

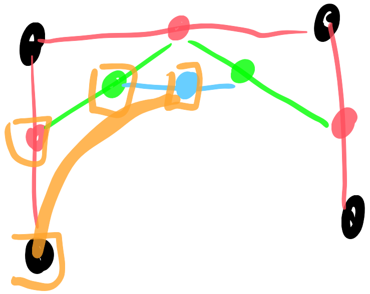


$$P_{12} = (1-v)P_1 + vP_2$$

Know the Destalejau Construction!

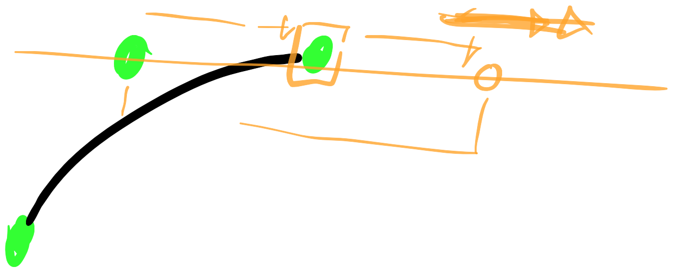
- helps with intuitions
- lets you compute values by hand
- useful for dividing curves





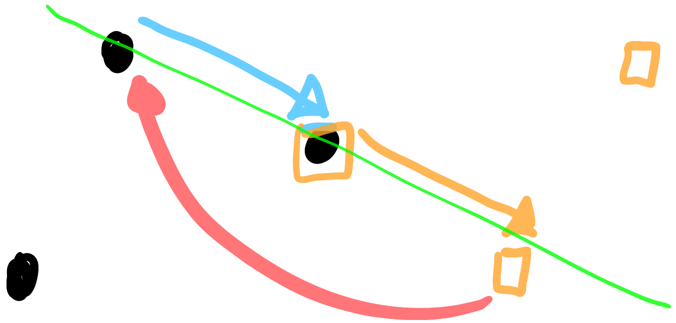
Designing with Bézier curves

APIs usually have cubics, often quadratics
(Canvas and SVG have both)



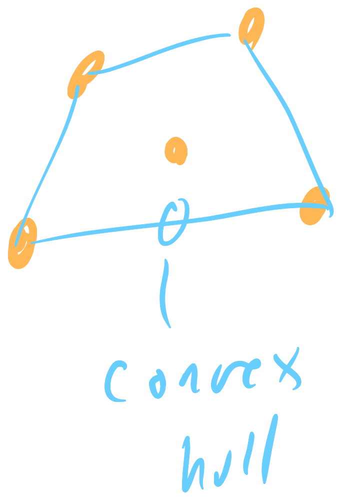
- C(0) continuity - match end points
- G(1) continuity - align interior points

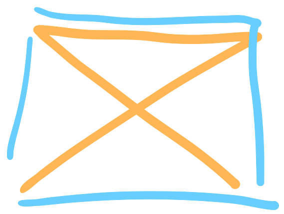
Each pieces is a polynomial (so it is continuous)



General Beziers

1. Interpolate the end points
2. Stay inside the ~~triangle~~ convex hull (polygon)
3. Not "wiggle too much" (variation diminishing - next slide)
4. Symmetry (forward/backwards)
5. Locality (only these points)
6. Control tangents
    - and higher derivatives

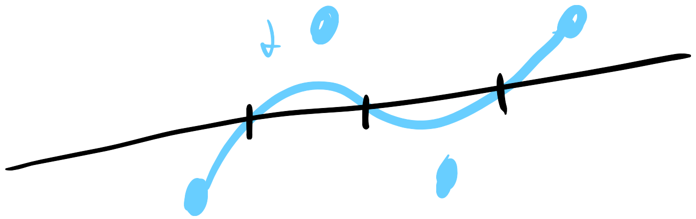


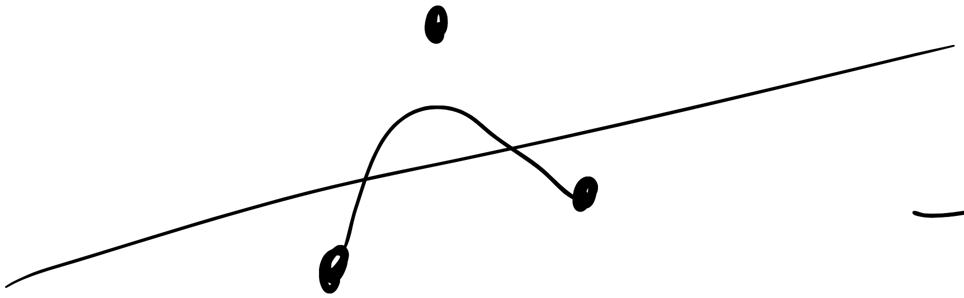


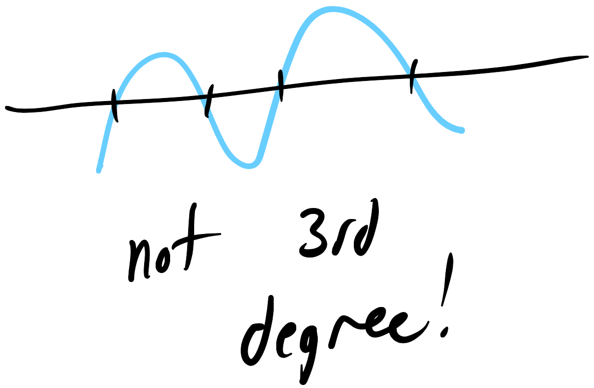
Variation diminishing

The wiggle theorem

The crossing property







Cubic Beziers (4 points)

The beginning tangent is 3x the vector $\mathbf{p_1} - \mathbf{p_0}$

The ending tangent is 3x the vector $\mathbf{p_3} - \mathbf{p_2}$

Similar to a Hermite

Stays inside the **convex hull**

Factor of 3 for tangents

Limited wiggles (max 3 crossings)

Equations?

Parametric equations can be derived (blending tree)

Have a nice form

Look them up when you need them

Cubics are cubics!

Canonical, Bezier, Cardinal, Hermite ....

Just different ways to describe the same curve (segment)

Convert between types

(APIs usually have Beziers)

Drawing Curves

- uniform steps in u
- non-uniform steps in u
- adaptive subdivision

More about Curves?

In Class

- Using this in practice (train project)
- Even steps (arc-length)
- Fancier Cardinals
- More on Beziers
- Getting smoother curves (C(2)) - B-Spline ideas
- How to choose curve types?

Probably Not in Class

- C(2) interpolation
- B-Spline details
- Bezier algorithms