# Parametric Curves

CSE 457

# Reading

Required:

- Angel 10.1-10.3, 10.5.2, 10.6-10.7, 10.9

Optional

- Bartels, Beatty, and Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*, 1987.
- Farin. *Curves and Surfaces for CAGD: A Practical Guide*, 4th ed., 1997.

# Curves before computers

The “loftsman’s spline”:

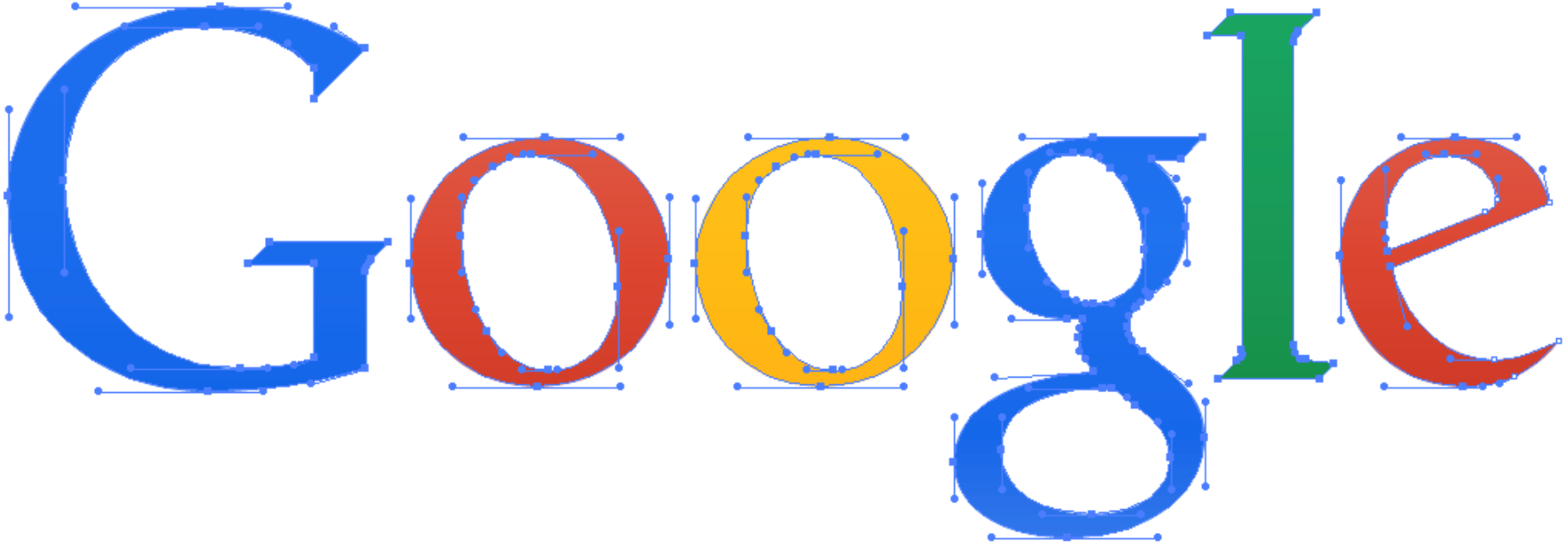
- long, narrow strip of wood or metal
- shaped by lead weights called “ducks”
- gives curves with second-order continuity, usually

Used for designing cars, ships, airplanes, etc.

But curves based on physical artifacts can’t be replicated well, since there’s no exact definition of what the curve is.

Around 1960, a lot of industrial designers were working on this problem.

Today, curves are easy to manipulate on a computer and are used for CAD, art, animation, …




Bezier curves… It has 100 anchor points, resulting in a 6 KB (6,380 bytes) file. When compressed, the size comes down to 2 KB (2,145 bytes).

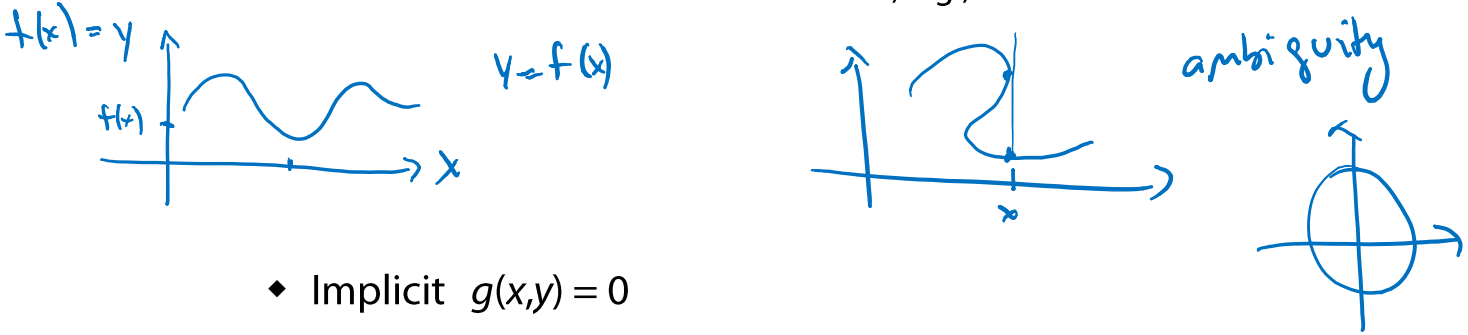
The entire logo consists of:

10 circles (2 each for the capital G and lower case g, 2 for each O, and 2 for the e)
5 rectangles (2 for the capital G, 1 for the lower case l, 2 for the e)
1 shape made with 7 anchor points (the descender on the lower-case g)

305 byte logo

# Mathematical curve representation

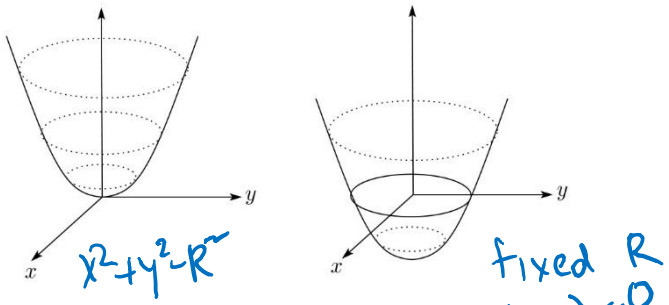
- Explicit $y=f(x)$
  - what if the curve isn't a function, e.g., a circle?




- Implicit $g(x,y) = 0$

$x^2+y^2 = R^2$

$g(x,y) = x^2+y^2-R^2$

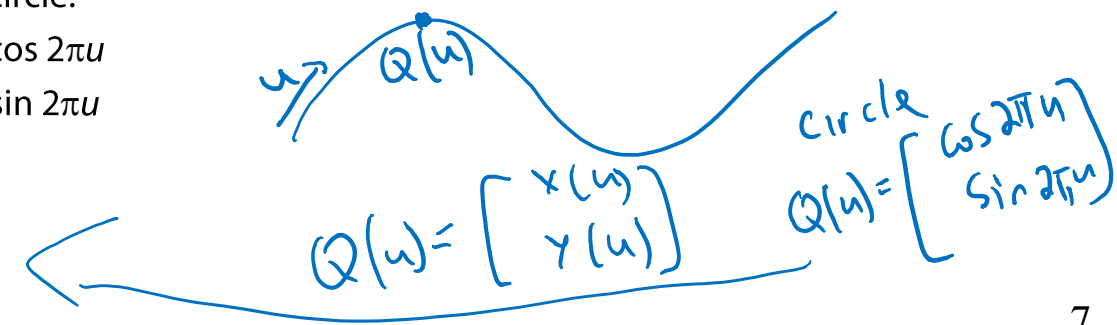



- Parametric $Q(u) = (x(u),y(u))$
  - For the circle:

    $x(u) = \cos 2\pi u$

    $y(u) = \sin 2\pi u$

$\cos^2 2\pi u + \sin^2 2\pi u = 1$



# Parametric polynomial curves

We'll use parametric curves, $Q(u)=(x(u),y(u))$, where the functions are all polynomials in the parameter.

polynomials:

$$x(u)=\sum_{k=0}^{n} a_k u^k = a_0 + a_1 u + a_2 u^2 + a_3 u^3 + \cdots$$

$$y(u)=\sum_{k=0}^{n} b_k u^k = b_0 + b_1 u + b_2 u^2 + b_3 u^3 + \cdots$$

$x'(u) = a_1 + 2a_2 u + 3a_3 u^2 + \cdots$

$x''(u) = 2a_2 + 6a_3 u + \cdots$

$x'''(u) = 6a_3 + \cdots$

Advantages:

- easy (and efficient) to compute
- ~~infinitely~~ differentiable (all derivatives above the $n^{th}$ derivative are zero)

We'll also assume that $u$ varies from 0 to 1. $0 \le u \le 1$

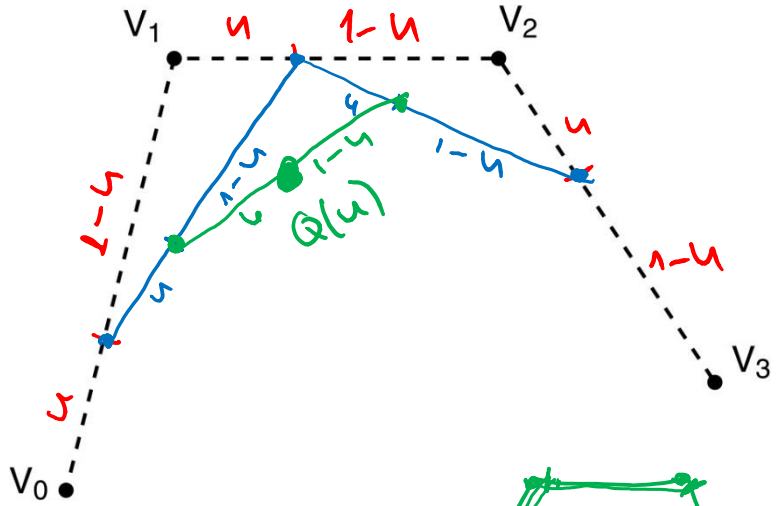
Note that we'll focus on 2D curves, but the generalization to 3D curves is completely straightforward.

# de Casteljau's algorithm

geometric way to create a curve out of 4 pts.

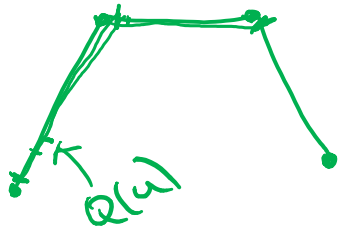
$V_0, V_1, V_2, V_3$ control polygon

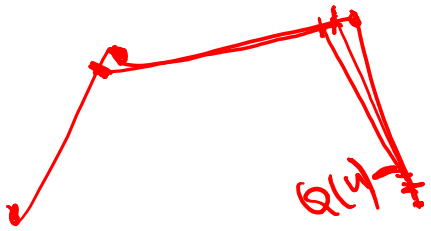
Recursive interpolation:




What if $u$=0? $Q(0) = V_0$

What if $u$=1? $Q(1) = V_3$

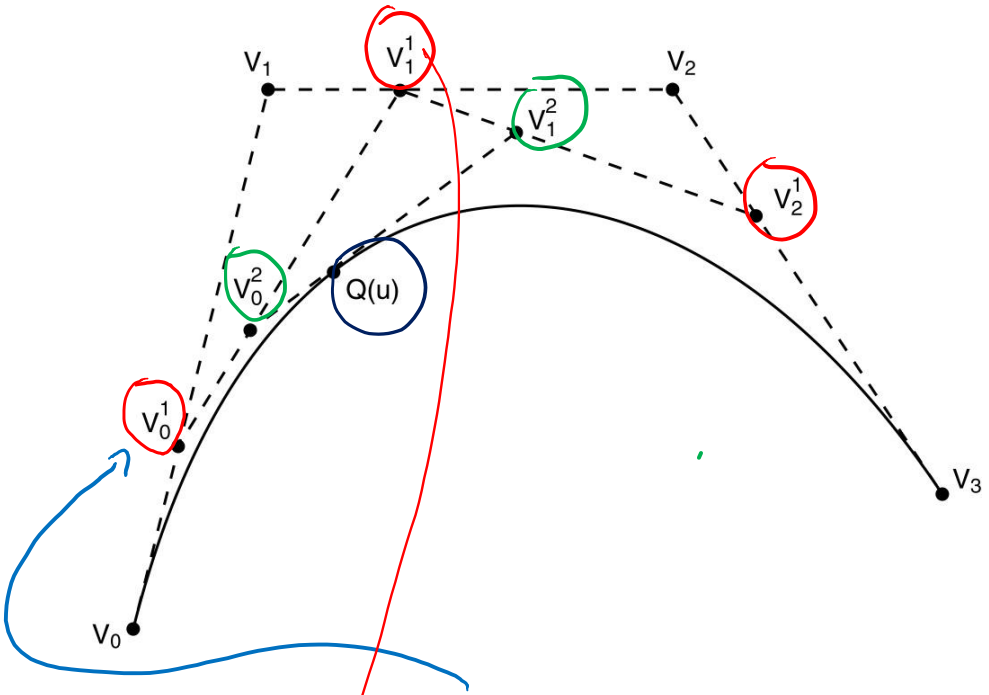


# de Casteljau's algorithm, cont'd

Recursive notation:

level 1 :
level 2
level 3




What is the equation for $V_0^1$ ?

$V_0' = V_0 + u(V_1 - V_0)$

$V_1' = V_1 + u(V_2 - V_1)$

$V_2' = V_2 + u(V_3 - V_2)$

$V_0^2 = V_0' + u(V_1' - V_0')$

$V_1^2 = V_1' + u(V_2' - V_1')$

$Q(u) = V_0^2 + u(V_1^2 - V_0^2)$

$= uV_1^2 + (1-u)V_0^2$

# Finding Q(u)

Let's solve for **Q**(u):

level 1

$$V_0^1 = (1-u)V_0 + uV_1$$
$$V_1^1 = (1-u)V_1 + uV_2$$
$$V_2^1 = (1-u)V_2 + uV_3$$

level 2

$$V_0^2 = (1-u)V_0^1 + uV_1^1$$
$$V_1^2 = (1-u)V_1^1 + uV_2^1$$

final

$$Q(u) = (1-u)V_0^2 + uV_1^2$$
$$= (1-u)[(1-u)V_0^1 + uV_1^1] + u[(1-u)V_1^1 + uV_2^1]$$
$$= (1-u)[(1-u)\{(1-u)V_0 + uV_1\} + u\{(1-u)V_1 + uV_2\}] + \ldots$$
$$Q(u) = (1-u)^3 V_0 + 3u(1-u)^2 V_1 + 3u^2(1-u)V_2 + u^3 V_3$$

3 cntrl pts

4 pts → degree 3 polynomial

# Finding Q(u) (cont'd)

In general,

$$Q(u) = \sum_{i=0}^{n} \binom{n}{i} u^i (1-u)^{n-i} V_i$$

where "*n* choose *i*" is:

$$\binom{n}{i} = \frac{n!}{(n-i)!i!}$$

This defines a class of curves called **Bézier curves**.

What's the relationship between the number of control points and the degree of the polynomials?

n+1 control points ⟶ n degree

# Bernstein polynomials

We can take the polynomial form:

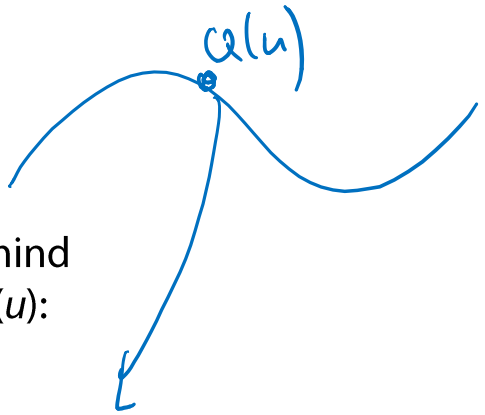
$$Q(u) = \sum_{i=0}^{n} \binom{n}{i} u^i (1-u)^{n-i} V_i$$

and re-write it as:

$$Q(u) = \sum_{i=0}^{n} b_i^n(u) V_i$$

where the $b_i$(u) are the **Bernstein polynomials**:

$$b_i^n(u) \equiv \binom{n}{i} u^i (1-u)^{n-i}$$

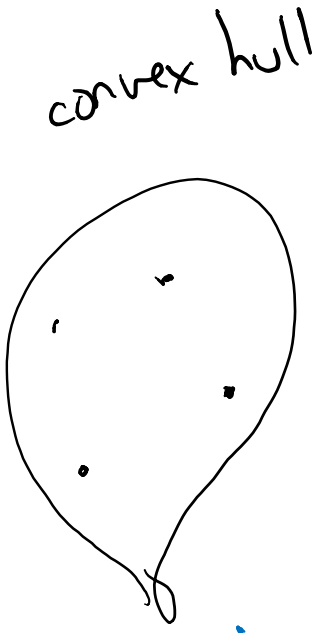



We can also expand the equation for $Q(u)$ to remind us that it is composed of polynomials $x(u)$ and $y(u)$:

$$Q(u) = \sum_{i=0}^{n} b_i^n(u) V_i = \sum_{i=0}^{n} b_i^n(u) \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} x_i b_i^n(u) \\ \sum_{i=0}^{n} y_i b_i^n(u) \\ \sum_{i=0}^{n} b_i^n(u) \end{bmatrix} = \begin{bmatrix} x(u) \\ y(u) \\ 1 \end{bmatrix}$$

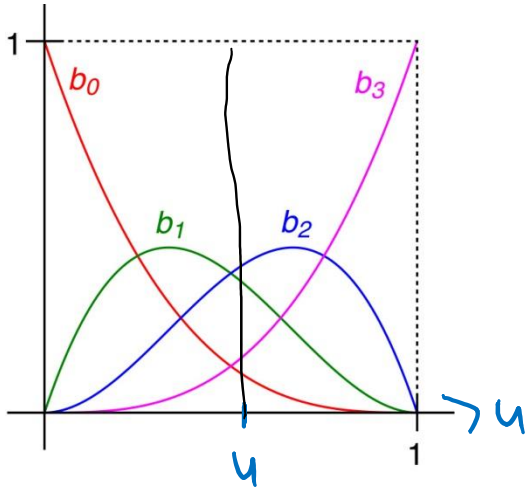
homogenous coordinates

# Bernstein polynomials, cont'd




For degree 3, the Bernstein polynomials are:

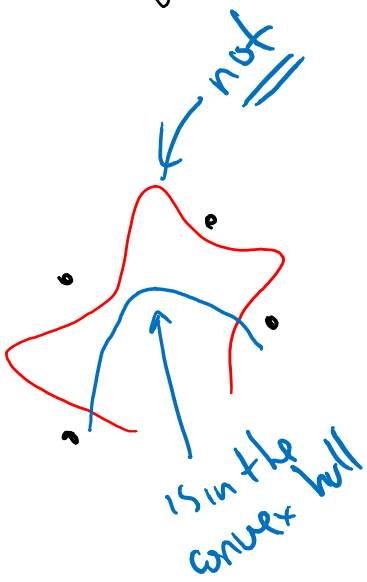
$$b_0^3(u) = (1-u)^3$$
$$b_1^3(u) = 3u(1-u)^2$$
$$b_2^3(u) = 3u^2(1-u)$$
$$b_3^3(u) = u^3$$




Useful properties (for Bernstein polynomials of any degree) on the interval [0,1]:

1. The sum of all four is exactly 1 for any *u*. (We say the curves form a "partition of unity").
2. Each polynomial has value between 0 and 1.

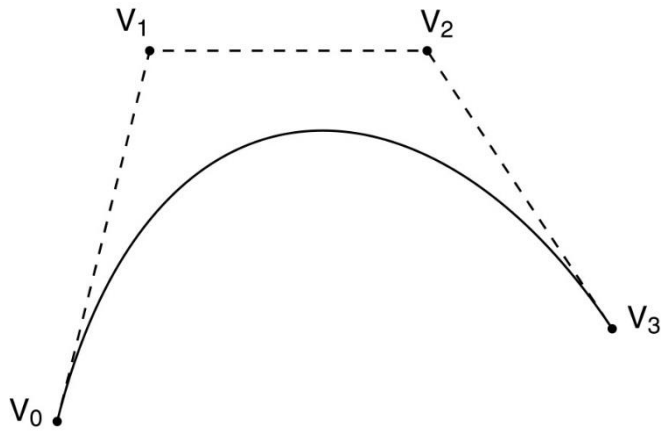
These together imply that the curve is generated by **convex combinations** of the control points and therefore lies within the **convex hull** of those control points.




The convex hull of a point set is the smallest convex polygon (in 2D) or polyhedron (in 3D) enclosing the points. In 2D, think of a string looped around the outside of the point set and then pulled tightly around the set.

# Displaying Bézier curves

How could we draw one of these things?




1) Select $u = 0, 0.01, 0.02, \ldots, 1$

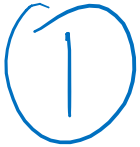
2) evaluate the poly. $b_i$

3) multiply by $v_i$ sum

# Curve desiderata

Bézier curves offer a fairly simple way to model parametric curves.

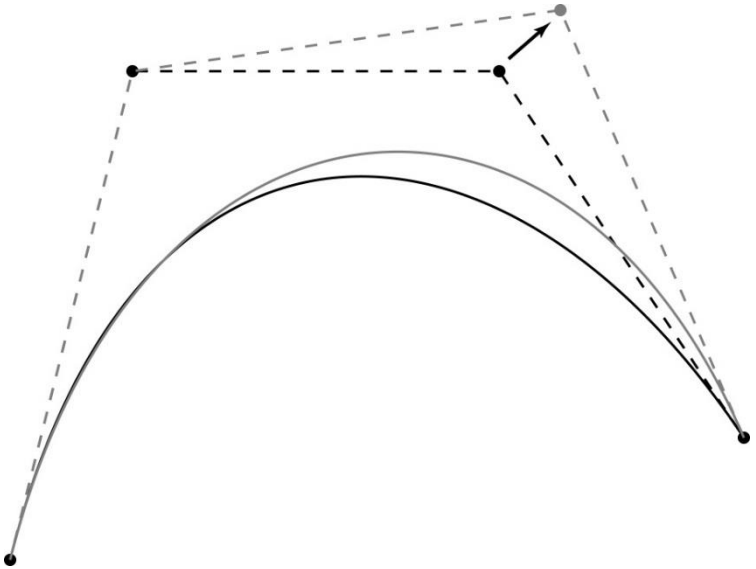
But, let's consider some general properties we would like curves to have…



# Local control

One problem with Béziers is that every control point affects every point on the curve (except the endpoints).

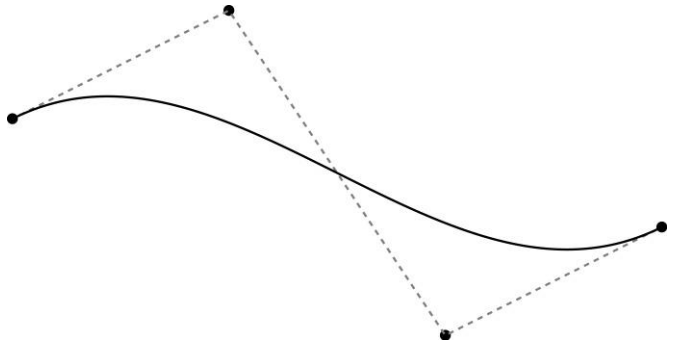
Moving a single control point affects the whole curve!



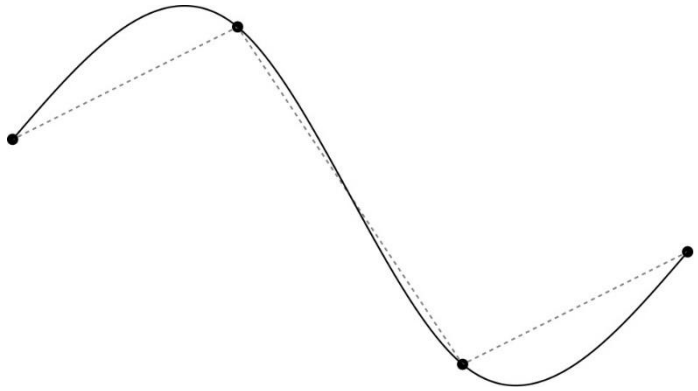
We'd like to have **local control**, that is, have each control point affect some well-defined neighborhood around that point.

# Interpolation

Bézier curves are **approximating**. The curve does not (necessarily) pass through all the control points. Each point pulls the curve toward it, but other points are pulling as well.



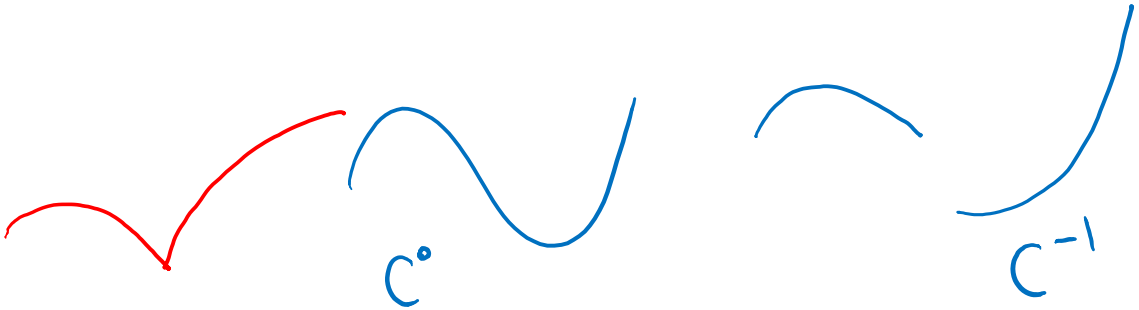
We'd like to have a curve that is **interpolating**, that is, that always passes through every control point.



# Continuity

We want our curve to have **continuity**: there shouldn't be any abrupt changes as we move along the curve.

"$0^{th}$ order" continuity would mean that curve doesn't jump from one place to another.




We can also look at derivatives of the curve to get higher order continuity.

# 1st and 2nd Derivative Continuity

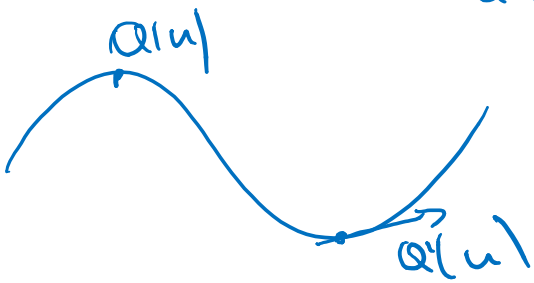
$C^1$ $C^2$

First order continuity implies continuous first derivative:

$C^1$: $$Q'(u) = \frac{dQ(u)}{du}$$

Let's think of u as "time" and $Q(u)$ as the path of a particle through space. What is the meaning of the first derivative, and which way does it point?

$Q'(u)$ velocity




Second order continuity means continuous second derivative:

$C^2$: $$Q''(u) = \frac{d^2Q(u)}{du^2}$$

What is the intuitive meaning of this derivative?

$Q''(u)$ acceleration

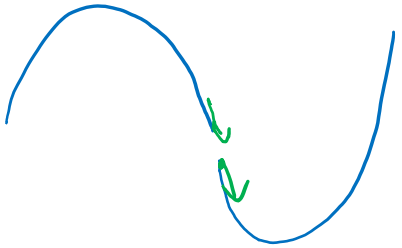
# $C^n$ (Parametric) Continuity

In general, we define $C^n$ continuity as follows:

$Q(u)$ is $C^n$ continuous

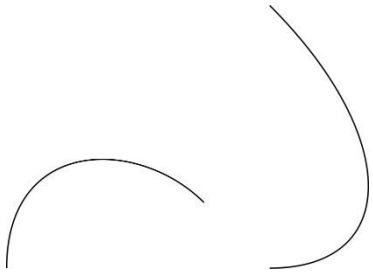
iff

$$Q^{(i)}(u) = \frac{d^i Q(u)}{du^i} \text{ is continuous for } 0 \le i \le n$$

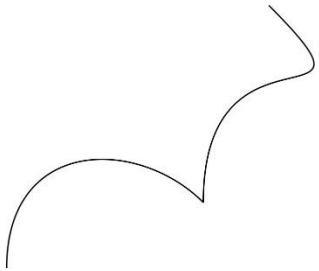


Note: these are *nested* degrees of continuity:

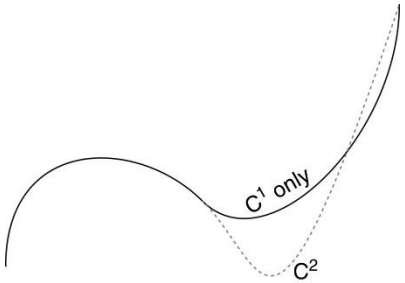
$C^{-1}$:



$C^0$:



$C^1$, $C^2$:




$C^3$, $C^4$, …:

Q'(u) cont. but it's not C' because of the jump and not $C^0$

# Bézier curves → splines

Bézier curves have C-infinity continuity on their interiors, but we saw that they do not exhibit local control or interpolate their control points.

It is possible to define points that we want to interpolate, and then solve for the Bézier control points that will do the job.

But, you will need as many control points as interpolated points -> high order polynomials -> wiggly curves. (And you still won't have local control.)

Instead, we'll splice together a curve from individual Béziers segments, in particular, cubic Béziers.

We call these curves **splines**.

The primary concern when splicing cuves together is getting good continuity at the endpoints where they meet…

# Local control

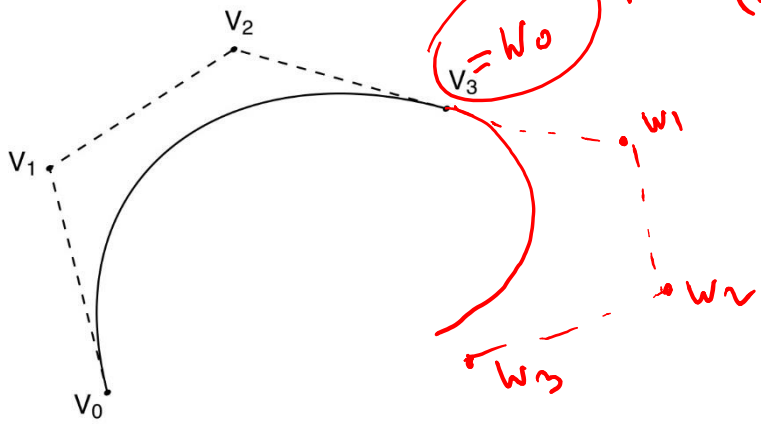
http://demonstrations.wolfram.com/InterpolatingASetOfData/

# Ensuring $C^0$ continuity

Suppose we have a cubic Bézier defined by $(V_0,V_1,V_2,V_3)$, and we want to attach another curve $(W_0,W_1,W_2,W_3)$ to it, so that there is $C^0$ continuity at the joint.

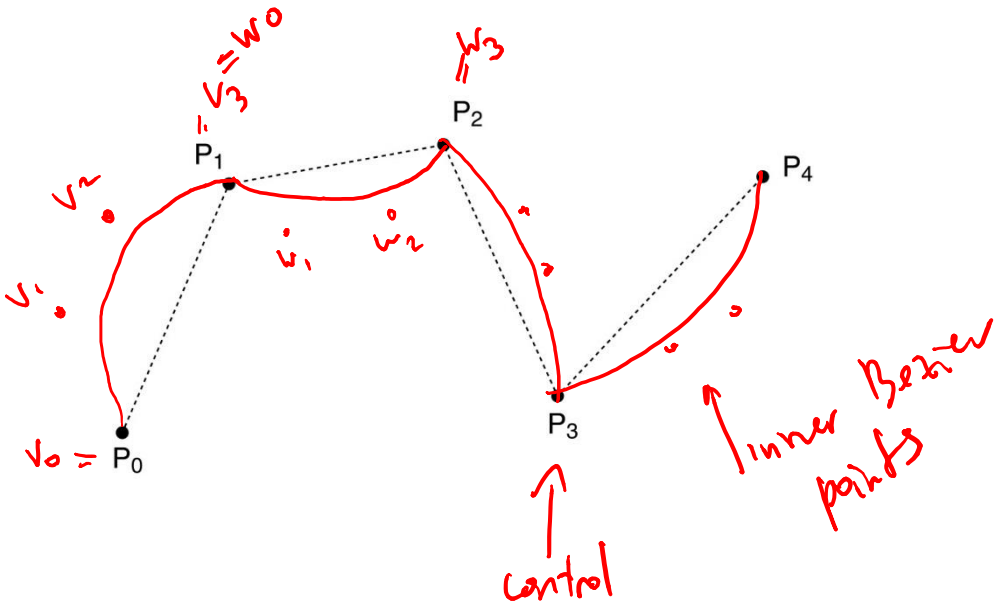
$$C^0 : Q_V(1) = Q_W(0)$$

What constraint(s) does this place on $(W_0,W_1,W_2,W_3)$?



# The $C^0$ Bezier spline

How then could we construct a curve passing through a set of points $P_1 \ldots P_n$?




We call this curve a **spline**. The endpoints of the Bezier segments are called **joints**. All other Bezier points (i.e., not endpoints) are called **inner Bezier points**; these points are generally not interpolated.

In the animator project, you will construct such a curve by specifying all the Bezier control points directly.

# 1st derivatives at the endpoints

For degree 3 (cubic) curves, we have already shown that we get:

$$Q(u) = (1-u)^3 V_0 + 3u(1-u)^2 V_1 + 3u^2(1-u)V_2 + u^3 V_3$$

We can expand the terms in $u$ and rearrange to get:

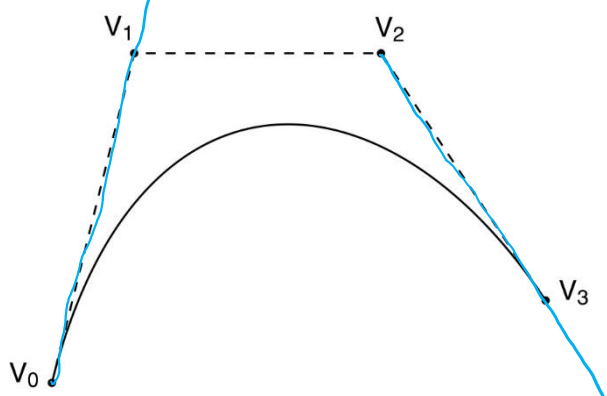
$$Q(u) = (-V_0 + 3V_1 - 3V_2 + V_3)u^3 + (3V_0 - 6V_1 + 3V_2)u^2 + (-3V_0 + 3V_1)u + V_0$$

$$Q'(u) = 3u^2(-V_0 + 3V_1 - 3V_2 + V_3) + 2u(3V_0 - 6V_1 + 3V_2) + (-3V_0 + 3V_1)$$

What then is the first derivative when evaluated at each endpoint, $u$=0 and $u$=1?

$Q'(0) = 3(V_1 - V_0)$

$Q'(1) = 3(V_3 - V_2)$



# Ensuring $C^1$ continuity

Suppose we have a cubic Bézier defined by $(V_0,V_1,V_2,V_3)$, and we want to attach another curve $(W_0,W_1,W_2,W_3)$ to it, so that there is $C^1$ continuity at the joint.

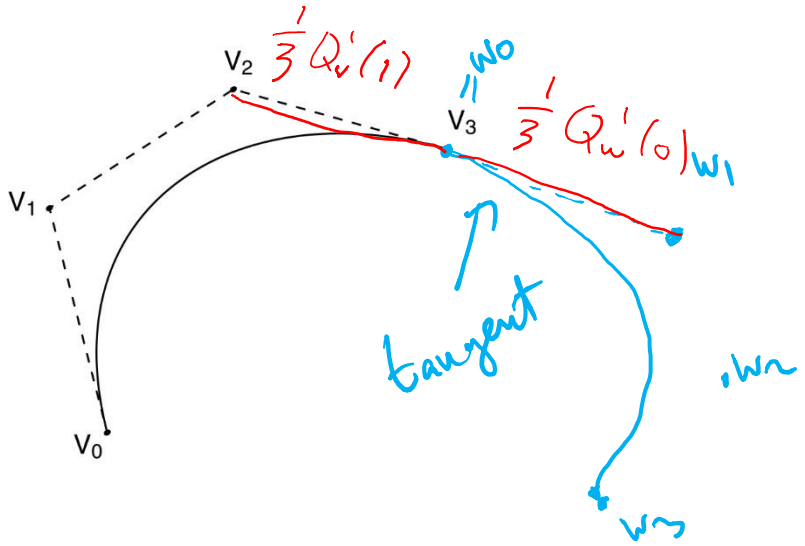
$$C^1: \begin{cases} Q_V(1) = Q_W(0) \\ Q_V'(1) = Q_W'(0) \end{cases}$$

+ $C^0$

new constraint.

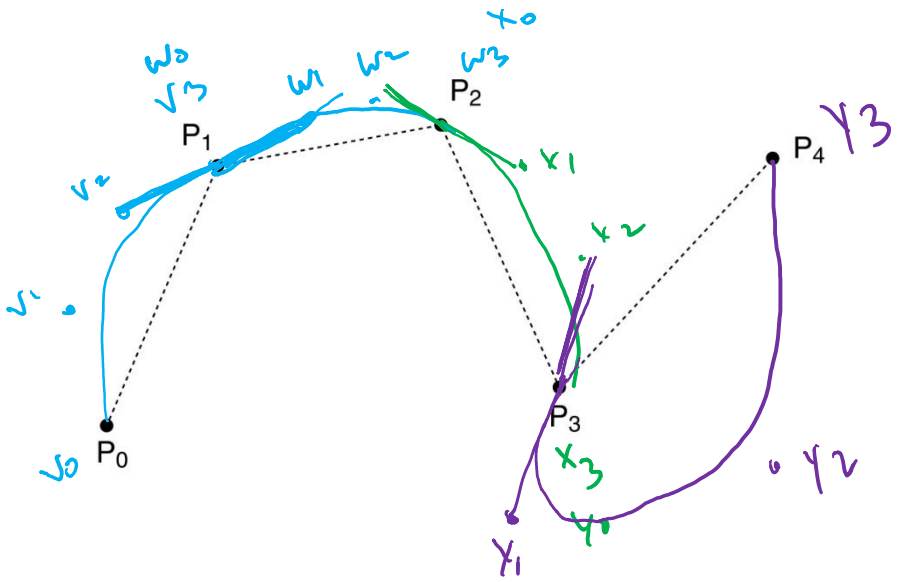
$V_3 - V_2$ $\quad$ $W_1 - W_0$

What constraint(s) does this place on $(W_0,W_1,W_2,W_3)$?



# The $C^1$ Bezier spline

How then could we construct a curve passing through a set of points $P_0 \ldots P_n$?



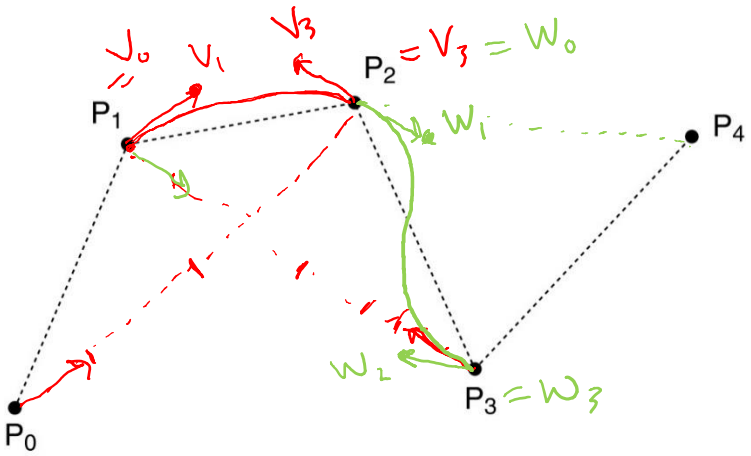

We can specify the Bezier control points directly, or we can devise a scheme for placing them automatically…

# Catmull-Rom splines

If we set each derivative to be one half of the vector between the previous and next controls, we get a **Catmull-Rom spline**.

This leads to:

$$V_0 = P_1$$
$$V_1 = P_1 + \frac{1}{6}(P_2 - P_0)$$
$$V_2 = P_2 - \frac{1}{6}(P_3 - P_1)$$
$$V_3 = P_2$$



# Catmull-Rom to Beziers

We can write the Catmull-Rom to Bezier transformation as:

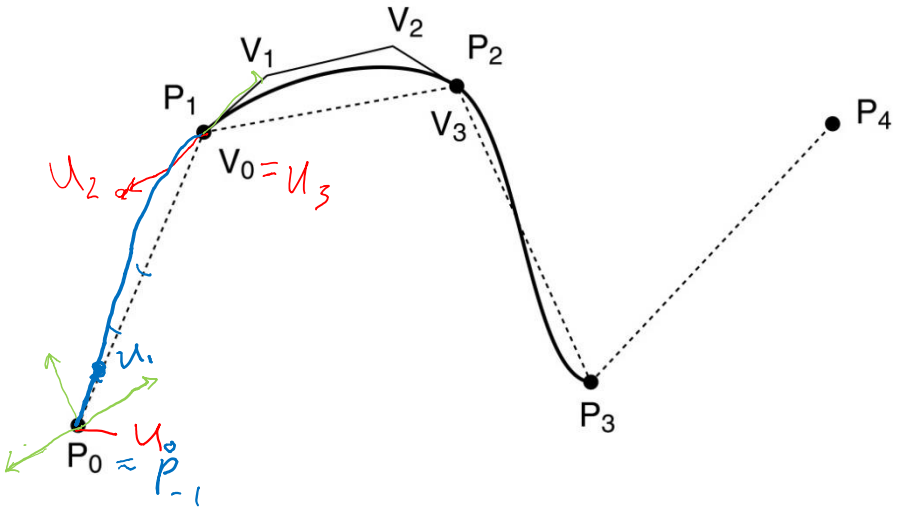
$$\begin{bmatrix} V_0^T \\ V_1^T \\ V_2^T \\ V_3^T \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1/6 & 1 & 1/6 & 0 \\ 0 & 1/6 & 1 & -1/6 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_0^T \\ P_1^T \\ P_2^T \\ P_3^T \end{bmatrix}$$

$$\mathbf{V} = \mathbf{M}_{\text{Catmull-Rom}}\mathbf{P}$$

# Endpoints of Catmull-Rom splines

We can see that Catmull-Rom splines don't interpolate the first and last control points.

By repeating those control points, we can force interpolation.

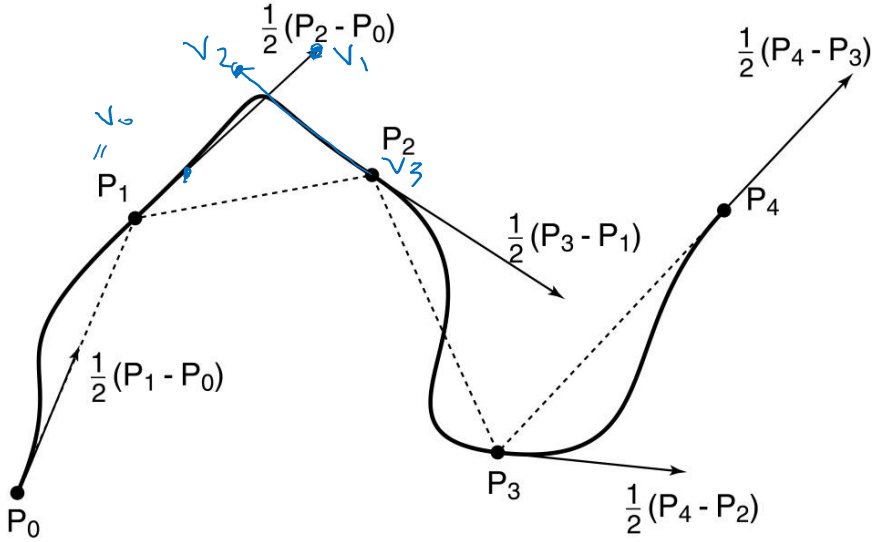


# Tension control

We can give more control by exposing the derivative scale factor as a parameter:

$$V_0 = P_1$$
$$V_1 = P_1 + \frac{\tau}{3}(P_2 - P_0)$$
$$V_2 = P_2 - \frac{\tau}{3}(P_3 - P_1)$$
$$V_3 = P_2$$

The parameter *t* controls the tension. Catmull-Rom uses *t* = 1/2. Here's an example with *t* =3/2.



# 2nd derivatives at the endpoints

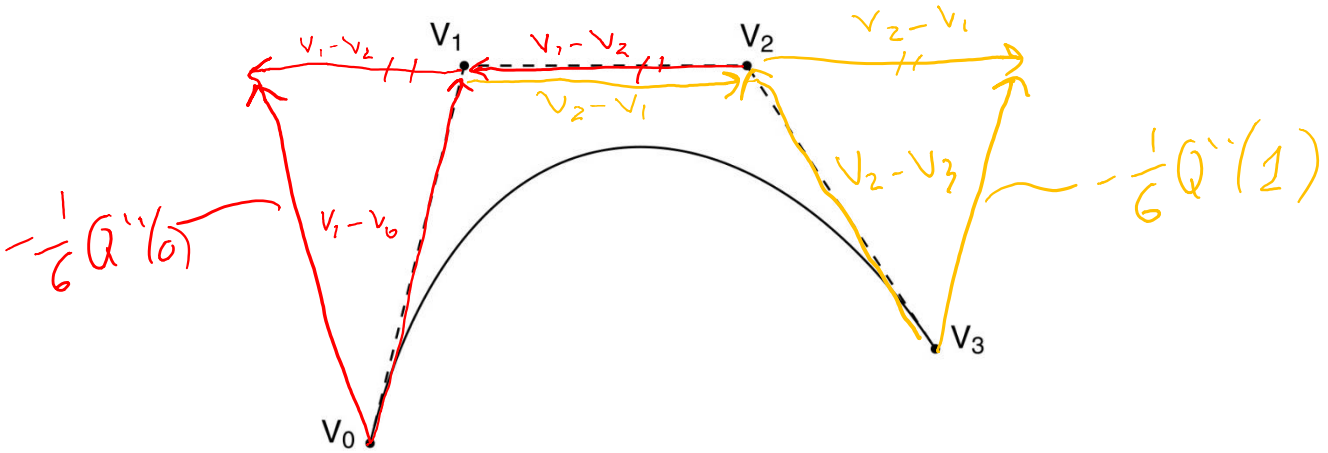
Finally, we'll want to develop $C^2$ splines. To do this, we'll need second derivatives of Bezier curves.

Taking the second derivative of $Q(u)$ yields:

$$\begin{aligned} Q''(0) &= 6(V_0 - 2V_1 + V_2) \\ &= -6[(V_1 - V_0) + (V_1 - V_2)] \\ Q''(1) &= 6(V_1 - 2V_2 + V_3) \\ &= -6[(V_2 - V_3) + (V_2 - V_1)] \end{aligned}$$

$-\frac{1}{6}Q''(0) = (V_1 - V_0) + (V_1 - V_2)$

$-\frac{1}{6}Q''(1) = (V_2 - V_3) + (V_2 - V_1)$



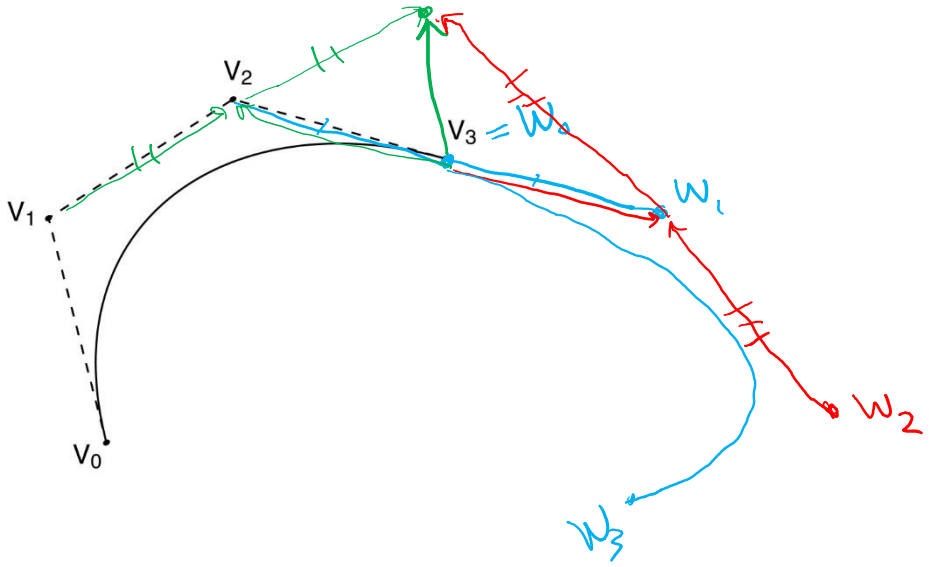
# Ensuring $C^2$ continuity

Suppose we have a cubic Bézier defined by $(V_0,V_1,V_2,V_3)$, and we want to attach another curve $(W_0,W_1,W_2,W_3)$ to it, so that there is $C^2$ continuity at the joint.

$$C^2 : \begin{cases} Q_V(1) = Q_W(0) \\ Q_V'(1) = Q_W'(0) \\ Q_V''(1) = Q_W''(0) \end{cases}$$

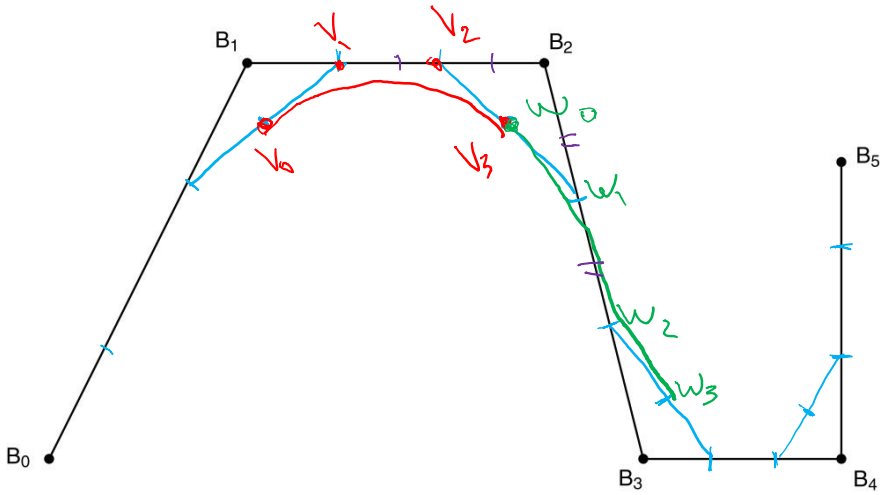
$V_3 = W_0$

What constraint(s) does this place on $(W_0,W_1,W_2,W_3)$?



# Building a complex spline

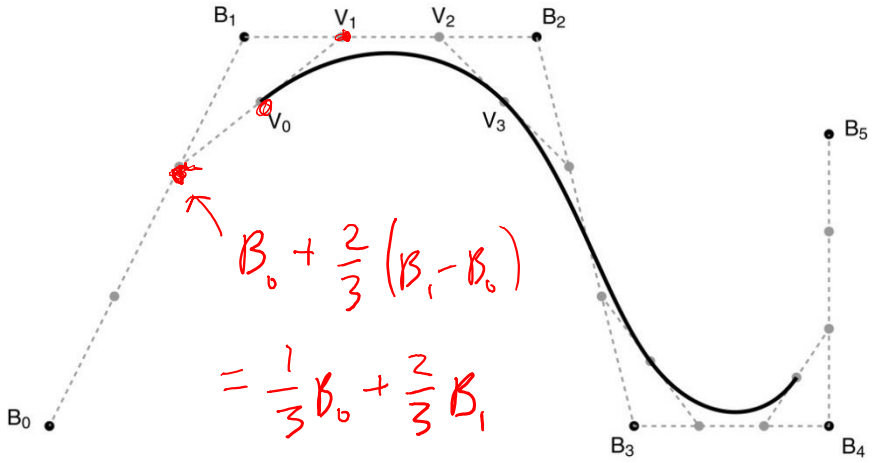
Instead of specifying the Bézier control points themselves, let's specify the corners of the A-frames in order to build a $C^2$ continuous spline.




These are called **B-splines**. The starting set of points are called **de Boor points**.

# B-splines

Here is the completed B-spline.




What are the Bézier control points, in terms of the de Boor points?

$$V_0 = \tfrac{1}{2}\left[\tfrac{1}{3}B_0 + \tfrac{2}{3}B_1\right] + \tfrac{1}{2}\left[\tfrac{2}{3}B_1 + \tfrac{1}{3}B_2\right] = \tfrac{1}{6}B_0 + \tfrac{2}{3}B_1 + \tfrac{1}{6}B_2$$

$$V_1 = \tfrac{2}{3}B_1 + \tfrac{1}{3}B_2$$

$$V_2 = \tfrac{1}{3}B_1 + \tfrac{2}{3}B_2$$

$$V_3 = \tfrac{1}{6}B_1 + \tfrac{2}{3}B_2 + \tfrac{1}{6}B_3$$

# B-splines to Beziers

We can write the B-spline to Bezier transformation as:

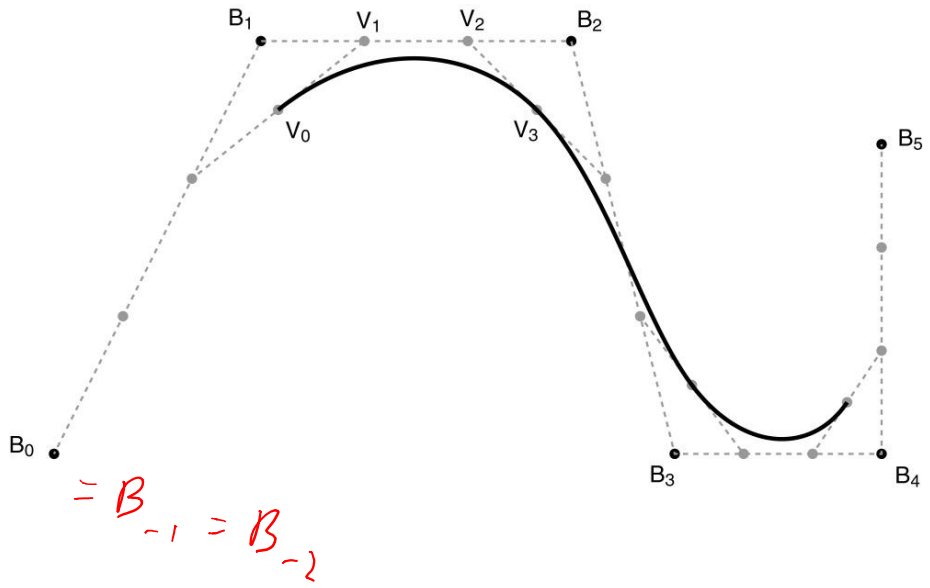
$$\begin{bmatrix} V_0^T \\ V_1^T \\ V_2^T \\ V_3^T \end{bmatrix} = \begin{bmatrix} 1/6 & 2/3 & 1/6 & 0 \\ 0 & 2/3 & 1/3 & 0 \\ 0 & 1/3 & 2/3 & 0 \\ 0 & 1/6 & 2/3 & 1/6 \end{bmatrix} \begin{bmatrix} B_0^T \\ B_1^T \\ B_2^T \\ B_3^T \end{bmatrix}$$

$$\mathbf{V} = \mathbf{M}_{\text{B-spline}}\mathbf{B}$$

# Endpoints of B-splines

As with Catmull-Rom splines, the first and last control points of B-splines are generally not interpolated.

Again, we can force interpolation by repeating the endpoints…*twice*.

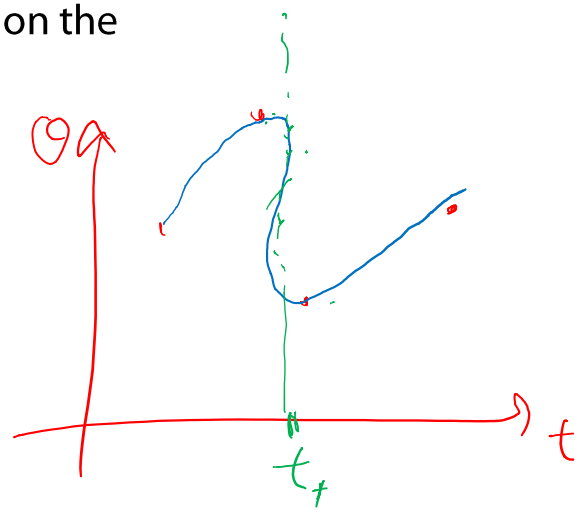



triple endpoints.

# Curves in the animator project

In the animator project, you will draw a curve on the screen:

$$\mathbf{Q}(u) = (x(u), y(u))$$




You will actually treat this curve as:

$$\theta(u) = y(u)$$
$$t(u) = x(u)$$

Where $\theta$ is a variable you want to animate. We can think of the result as a function:

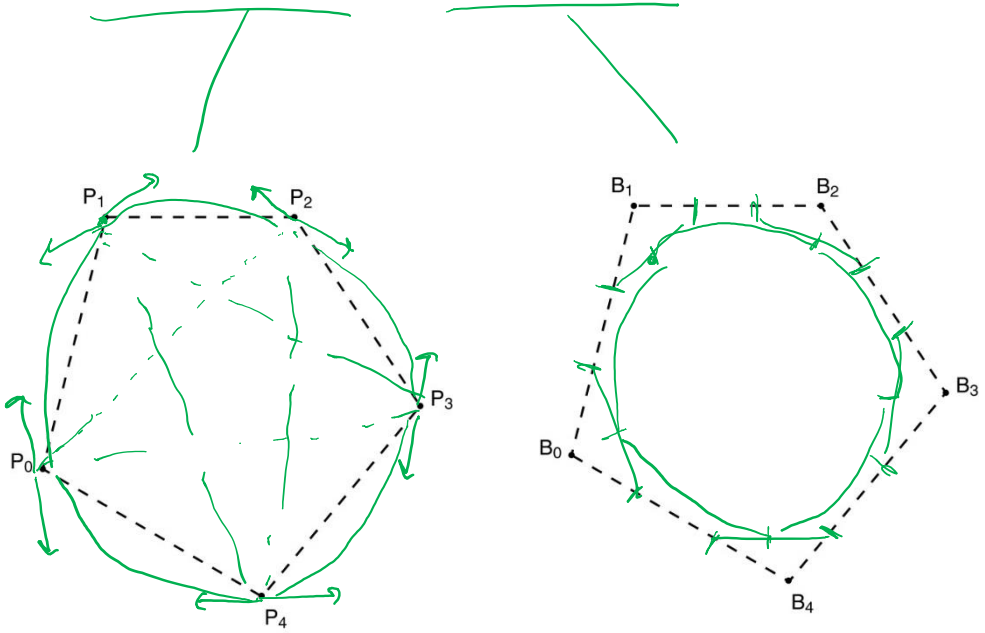
$$\theta(t)$$

In general, you have to apply some constraints to make sure that $\theta(t)$ actually is a *function*.

# Closing the loop

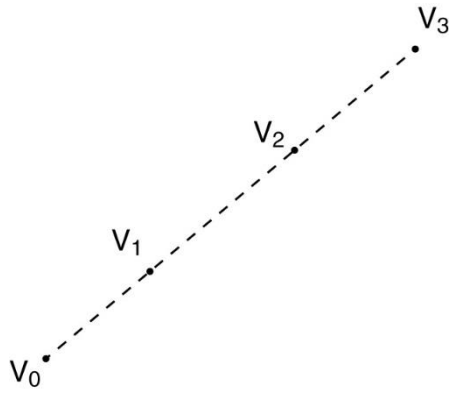
What if we want a closed curve, i.e., a loop?

With Catmull-Rom and B-spline curves, this is easy:



# Drawing Bézier curves, revisited

Let's return to the question of how to draw Bezier curves, the building block for splines. Consider a set of Bézier control points are arranged as follows:

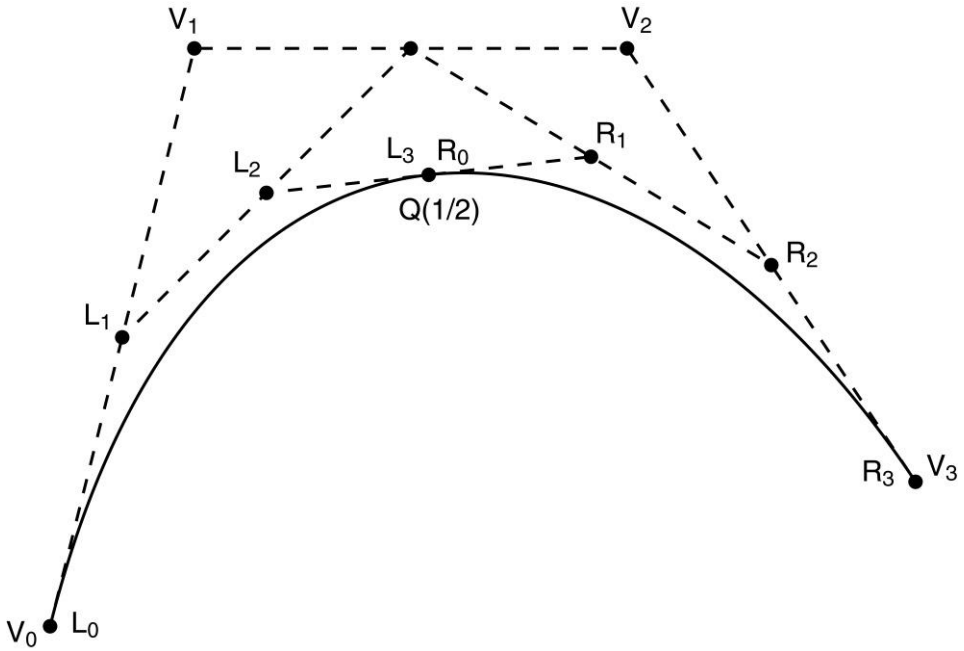


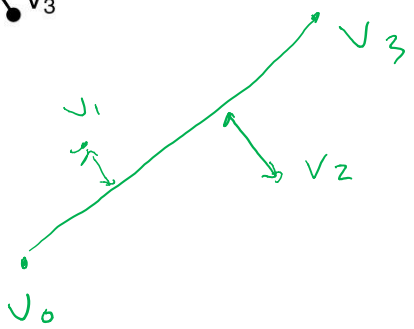

How many line segments do you really need to draw?

It would be nice if we had an *adaptive* algorithm, that would take into account flatness.

```
DisplayBezier( V0, V1, V2, V3 )
begin
    if ( FlatEnough( V0, V1, V2, V3 ) )
        Line( V0, V3 );
    else
        something;
end;
```

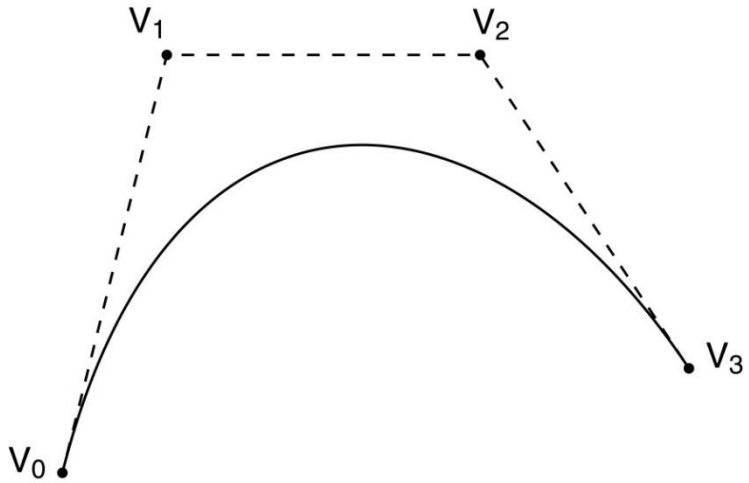
# Subdivide and conquer



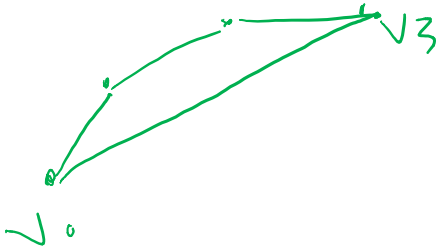




```
DisplayBezier( V0, V1, V2, V3 )
begin
    if ( FlatEnough( V0, V1, V2, V3 ) )
        Line( V0, V3 );
    else
        Subdivide(V[]) ⇒ L[], R[]
        DisplayBezier( L0, L1, L2, L3 );
        DisplayBezier( R0, R1, R2, R3 );
end;
```

# Testing for flatness




Compare total length of control polygon to length of line connecting endpoints:




$$\frac{|V_0 - V_1| + |V_1 - V_2| + |V_2 - V_3|}{|V_0 - V_3|} < 1 + \varepsilon$$

# Reparameterization

We have so far been considering parametric continuity, derivatives w.r.t. the parameter *u*.

This form of continuity makes sense particularly if we really are describing a particle moving over time and want its motion (e.g., velocity and acceleration) to be smooth.

But, what if we're thinking only in terms of the shape of the curve? Is the parameterization actually intrinsic to the shape, i.e., is it the case that a shape has only one parameterization?

# Arc length parameterization

We can reparameterize a curve so that equal steps in parameter space (we'll call this new parameter "*s*") map to equal distances along the curve:

$$Q(s) \Rightarrow \Delta s = s_2 - s_1 = arclength\left[Q(s_1), Q(s_2)\right]$$

We call this an arc length parameterization. We can re-write the equal step requirement as:

$$\frac{arclength\left[Q(s_1), Q(s_2)\right]}{s_2 - s_1} = 1$$

Looking at very small steps, we find:

$$\lim_{s_2 \to s_1} \frac{arclength\left[Q(s_1), Q(s_2)\right]}{s_2 - s_1} = \left\| \frac{dQ(s)}{ds} \right\| = 1$$

# $G^n$ (Geometric) Continuity

Now, we define *geometric* $G^n$ continuity as follows:

$$Q(s) \text{ is } G^n \text{ continuous}$$
$$\text{iff}$$
$$Q^{(i)}(s) = \frac{d^i Q(s)}{ds^i} \text{ is continuous for } 0 \le i \le n$$

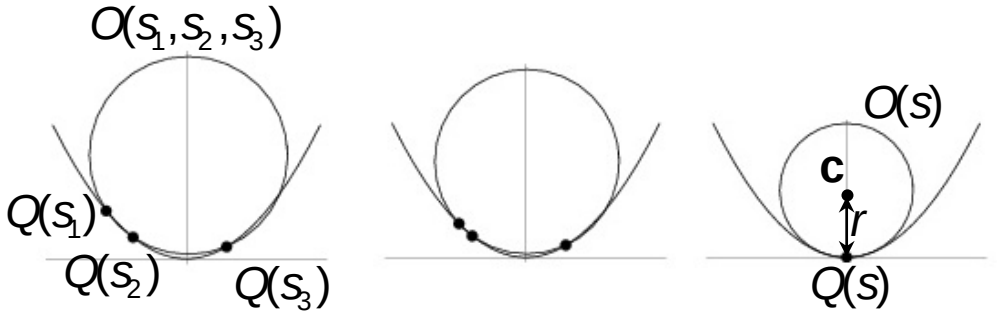
Where Q(*s*) is parameterized by arc length.

The first derivative still points along the tangent, but its length is always 1.

$G^n$ continuity is usually a weaker constraint than $C^n$ continuity (e.g., "speed" along the curve does not matter).

# $G^n$ Continuity (cont'd)

The second derivative now has a specific geometric interpretation. First, the "osculating circle" at a point on a curve can be defined based on the limit behavior of three points moving toward each other:




$$O(s) = \lim_{s_1, s_2, s_3 \to s} O(s_1, s_2, s_3)$$

The second derivative $Q''(s)$ then has these properties:

$$\|Q''(s)\| = \kappa(s) = \frac{1}{r(s)} \qquad Q''(s) = \mathbf{c}(s) - Q(s)$$

where $r(s)$ and $\mathbf{c}(s)$ are the radius and center of $O(s)$, respectively, and $k(s)$ is the "curvature" of the curve at $s$.

# Rational polynomial curves

Remarkably, parametric polynomial curves **cannot** represent something as simple as a circle!

BUT, ratios of polynomials can. We can write these in terms of homogeneous coordinates, which we then normalize:

$$Q(u) = \begin{bmatrix} \sum_{k=0}^{n} a_k u^k \\ \sum_{k=0}^{n} b_k u^k \\ \sum_{k=0}^{n} c_k u^k \end{bmatrix} \xrightarrow[\div \sum_{k=0}^{n} c_k u^k]{\text{Normalize}} \tilde{Q}(u) = \begin{bmatrix} \sum_{k=0}^{n} a_k u^k \Big/ \sum_{k=0}^{n} c_k u^k \\ \sum_{k=0}^{n} b_k u^k \Big/ \sum_{k=0}^{n} c_k u^k \\ 1 \end{bmatrix}$$

The equations above describe a **rational Bézier** curve.

It can be represented in terms of control points, but now we add the homogenous dimension. So for a 2D curve, we have control points with *three* components (lofted up into 3D), where the homogenous component can be something other than 1.

# Rational polynomial curves (cont'd)

What do we get for the following curve?

$$Q(u) = \begin{bmatrix} 2u \\ 1-u^2 \\ 1+u^2 \end{bmatrix}$$

**Q**: How does Illustrator represent a circle?

# NURBS

In general, we can spline together rational Bézier curves, to get things like **rational B-splines**.

Another thing we can do is vary the range of *u* so that it is not always [0..1] in each Bézier segment of a spline. E.g, it could be [0..1] in one segment and then [0..2] in the next.

The *u*-range affects placement of control points. The result is a **non-uniform** spline.

A very common type of spline is a **Non-Uniform Rational B-Spline** or **NURBS**.

(The "B" in B-spline technically stands for "Basis.")

# Summary

What to take home from this lecture:

- Geometric and algebraic definitions of Bézier curves.
- Basic properties of Bézier curves.
- How to display Bézier curves with line segments.
- Meanings of $C^k$ continuities.
- Geometric conditions for continuity of cubic splines.
- Properties of B-splines and Catmull-Rom splines.
- Geometric construction of B-splines and Catmull-Rom splines.
- How to construct closed loop splines.