Parametric curves

Brian Curless
CSE 557
Fall 2009

Reading

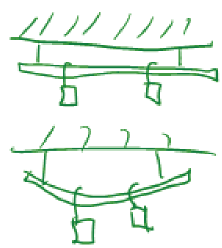
Required:

- Shirley 2.5, Chapter 15

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.

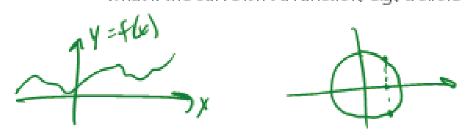
But curves based on physical artifacts can't be replicated well, since there's no exact definition of what the curve is.

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, …

Mathematical curve representation

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

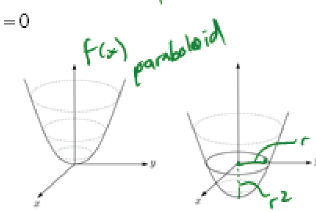


- Implicit $g(x,y) = 0$

$x^2+y^2=r^2$

$f(x,y)=x^2+y^2$

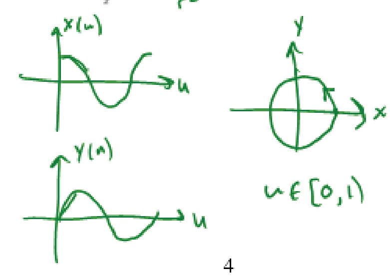
$g(x,y)=x^2+y^2-r^2$



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

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

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



$u \in [0,1]$

Parametric polynomial curves

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

$$x(u)=\sum_{k=0}^{n} a_k u^k$$

$$y(u)=\sum_{k=0}^{n} b_k u^k$$

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.

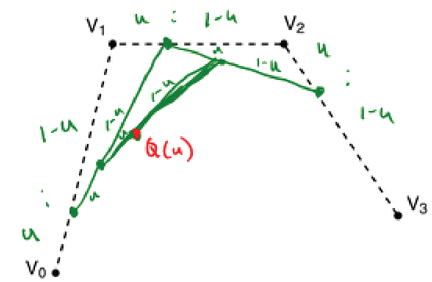
Note that we'll focus on 2D curves, but the generalization to 3D curves is completely straightforward.

de Casteljau's algorithm

Recursive interpolation:

$V_0 \cdots V_3$ control points

$[V_0 \cdots V_3]$ → control polygon



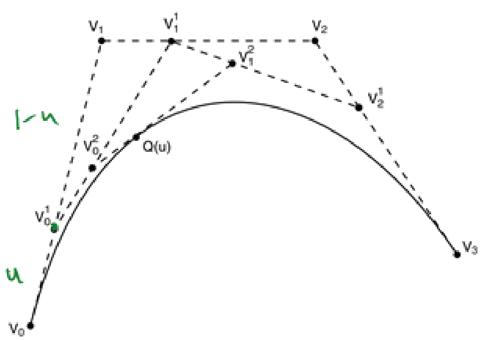
What if u=0? $Q(0) = V_0$

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

endpoint interpolating V_0, V_3

de Casteljau's algorithm, cont'd

Recursive notation:



What is the equation for V_0^1?

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

Finding Q(u)

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

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

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

$$\begin{aligned} Q(u) &= (1-u)V_0^2 + uV_1^2 \\ &= (1-u)[(1-u)V_0^1 + uV_1^1] + u[(1-u)V_1^1 + uV_2^1] \\ &= (1-u)[(1-u)\{(1-u)V_0 + uV_1\} + u\{(1-u)V_1 + uV_2\}] + \dots \\ &= (1-u)^3 V_0 + 3u(1-u)^2 V_1 + 3u^2(1-u)V_2 + u^3 V_3 \end{aligned}$$

$$= \begin{bmatrix} (1-u)^3 V_{0,x} + 3u(1-u^2)V_{1,x} + \cdots \\ (1-u)^3 V_{0,y} + \cdots \end{bmatrix}$$

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:

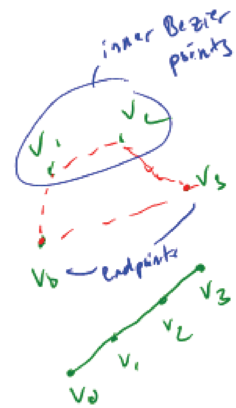
$$\binom{n}{i} = \frac{n!}{(n-i)!\,i!}$$

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^{th} degree $\Rightarrow$ $n+1$ ctrl. pts.

Bernstein polynomials

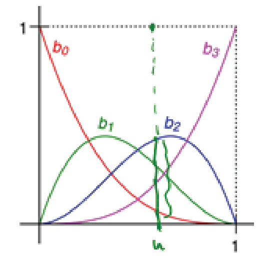


The coefficients of the control points are a set of functions called the **Bernstein polynomials:**

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

For degree 3, we have:

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



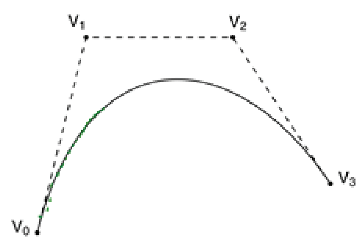
Useful properties on the interval [0,1]:

convex { affine {
- each polynomial has value between 0 and 1
- sum of all four is exactly 1 (a.k.a., a "partition of unity")

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.

Displaying Bézier curves

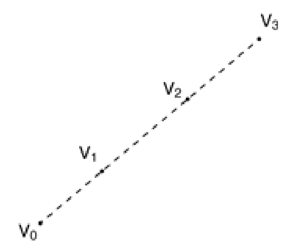
How could we draw one of these things?



$u = 0.0, 0.01, 0.02$ ...

Adaptive Sampling of Bézier curves

Suppose the control points are arranged as follows:

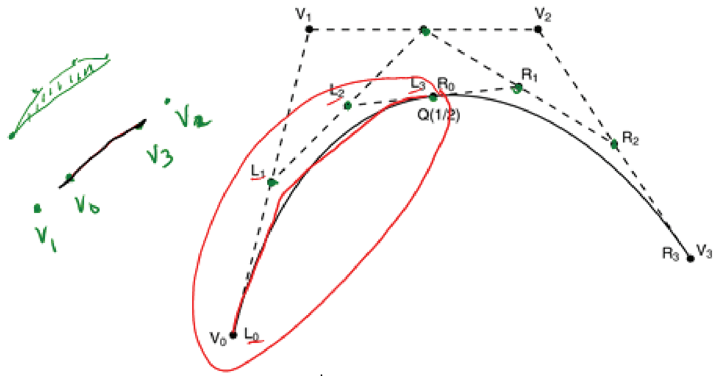


How many line segments do you really need to draw?

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

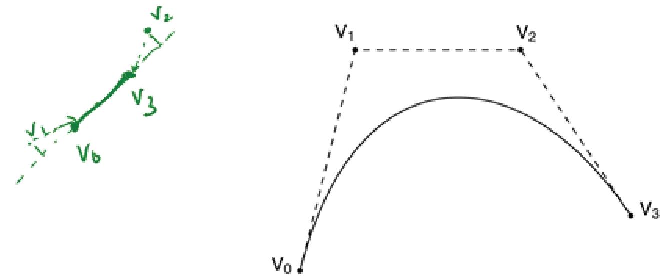
```
DisplayBezier( V0, V1, V2, V3 )
begin
    if ( FlatEnough( V0, V1, V2, V3 ) )
        Line( V0, V3 );
    else
        something;
end;
```

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:

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

Curve desiderata

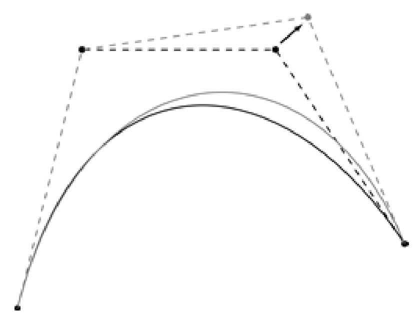
Bézier curves offer a fairly simple way to model parametric curves.

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).

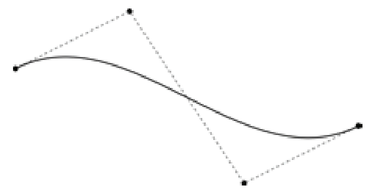
Moving a single control point affects the whole curve!



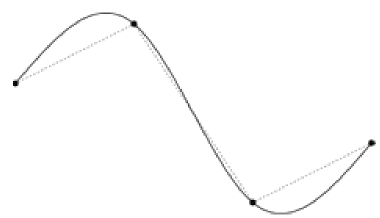
We'd like to have **local control**, that is, have each control point affect some well-defined neighborhood around that point.

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.



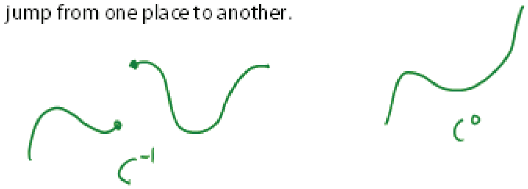
We'd like to have a curve that is **interpolating**, that is, that always passes through every control point.



Continuity

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

"0th 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.

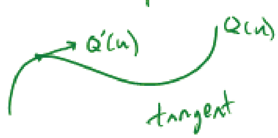
1st and 2nd Derivative Continuity

First order continuity implies continuous first derivative:

$$Q'(u) = \frac{dQ(u)}{du}$$

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?

velocity



Second order continuity means continuous second derivative:

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

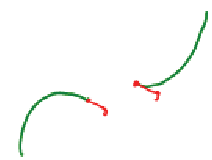
What is the intuitive meaning of this derivative?

acceleration

C^n (Parametric) Continuity

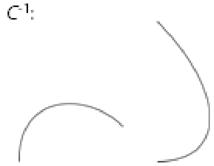
In general, we define C^n continuity as follows:

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

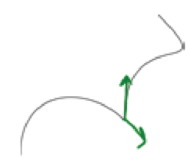


Note: these are *nested* degrees of continuity:

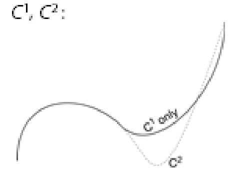
C^{-1}:



C^0:



C^1, C^2:



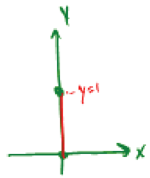
C^3, C^4, … :

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?

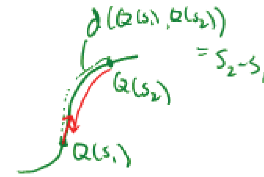


$$Q(u) = \begin{bmatrix} 0 \\ u \end{bmatrix} \qquad Q'(u) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\tilde{Q}(u) = \begin{bmatrix} 0 \\ u^2 \end{bmatrix} \qquad \tilde{Q}'(u) = \begin{bmatrix} 0 \\ u \end{bmatrix}$$

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

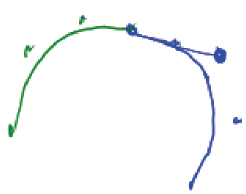
Now, we define *geometric* G^n continuity as follows:

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

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

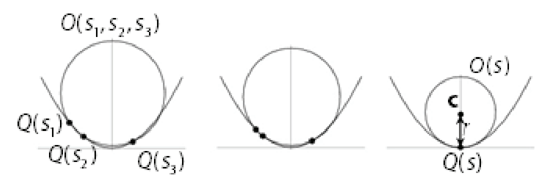
The first derivative still points along the tangent, but its length is always 1.

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:

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

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

We'll focus on C^n (i.e., parametric) continuity of curves for the remainder of this lecture.

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**.

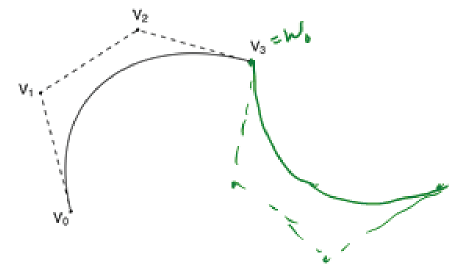
The primary concern when splicing cuves together is getting good continuity at the endpoints where they meet…

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.

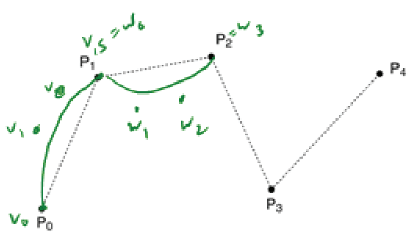
$$C^0 : Q_V(1) = Q_W(0)$$

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**.

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:

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

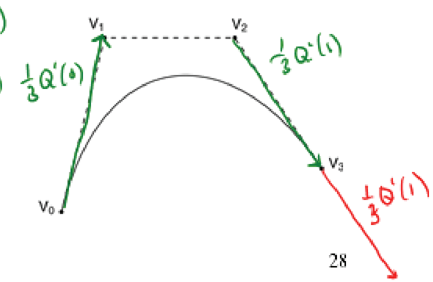
We can expand the terms in u and rearrange to get:

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

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

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

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

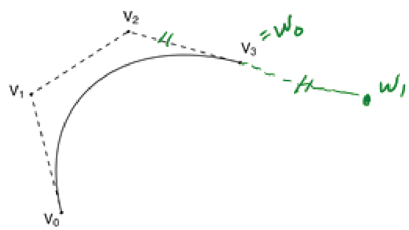


Ensuring C^1 continuity

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.

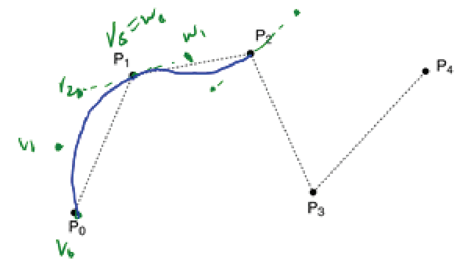
$$C^1 : \begin{cases} Q_V(1) = Q_W(0) \\ Q_V'(1) = Q_W'(0) \end{cases}$$

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 $P_0 \ldots P_n$?



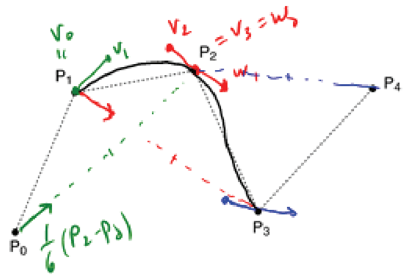
We can specify the Bezier control points directly, or we can devise a scheme for placing them automatically...

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:

$$\begin{aligned} V_0 &= P_1 \\ V_1 &= P_1 + \tfrac{1}{6}(P_2 - P_0) \\ V_2 &= P_2 - \tfrac{1}{6}(P_3 - P_1) \\ V_3 &= P_2 \end{aligned}$$



Catmull-Rom to Beziers

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

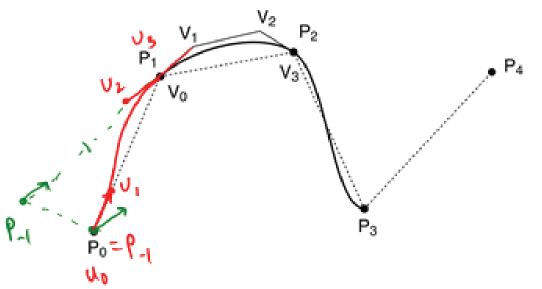
$$\begin{bmatrix} V_0^T \\ V_1^T \\ V_2^T \\ V_3^T \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1/6 & 1 & 1/6 & 0 \\ 0 & 1/6 & 1 & -1/6 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_0^T \\ P_1^T \\ P_2^T \\ P_3^T \end{bmatrix}$$

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

Endpoints of Catmull-Rom splines

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

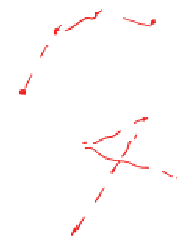
By repeating those control points, we can force interpolation.



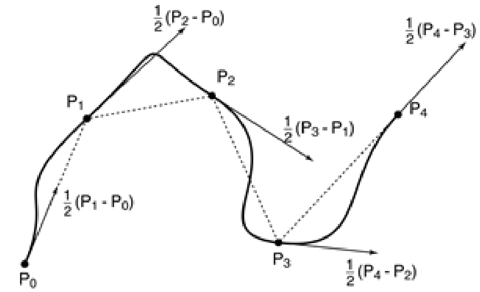
Tension control

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 τ controls the tension. Catmull-Rom uses $\tau = 1/2$. Here's an example with $\tau = 3/2$.

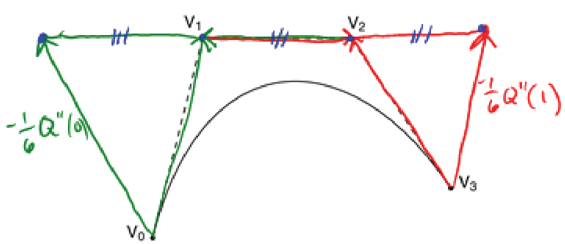


2nd derivatives at the endpoints

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:

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

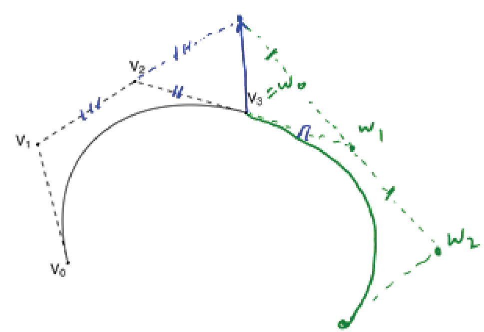


Ensuring C² continuity

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.

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

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

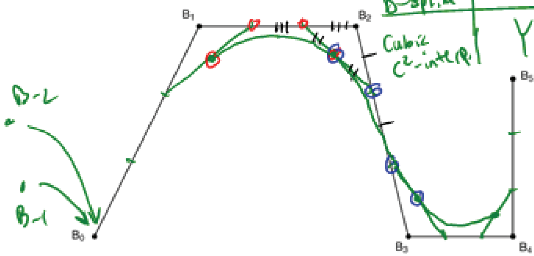


Building a complex spline

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.

Cubics

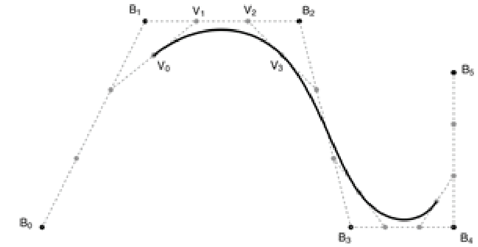
	interp.	C^2	local
C-R	Y	N	Y
B-spline	N	Y	Y
Cubic C^2-interp.	Y	Y	N



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?

$$\begin{aligned}
V_0 &= \underline{\quad}[\underline{\quad}B_0 + \underline{\quad}B_1] \\
&\quad + \underline{\quad}[\underline{\quad}B_1 + \underline{\quad}B_2] \\
&= \underline{\quad}B_0 + \underline{\quad}B_1 + \underline{\quad}B_2 \\
V_1 &= \underline{\quad}B_1 + \underline{\quad}B_2 \\
V_2 &= \underline{\quad}B_1 + \underline{\quad}B_2 \\
V_3 &= \underline{\quad}B_1 + \underline{\quad}B_2 + \underline{\quad}B_3
\end{aligned}$$

B-splines to Beziers

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

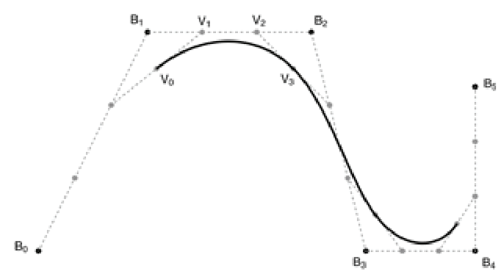
$$\begin{bmatrix} V_0^T \\ V_1^T \\ V_2^T \\ V_3^T \end{bmatrix} = \begin{bmatrix} 1/6 & 2/3 & 1/6 & 0 \\ 0 & 2/3 & 1/3 & 0 \\ 0 & 1/3 & 2/3 & 0 \\ 0 & 1/6 & 2/3 & 1/6 \end{bmatrix} \begin{bmatrix} B_0^T \\ B_1^T \\ B_2^T \\ B_3^T \end{bmatrix}$$

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

Endpoints of B-splines

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

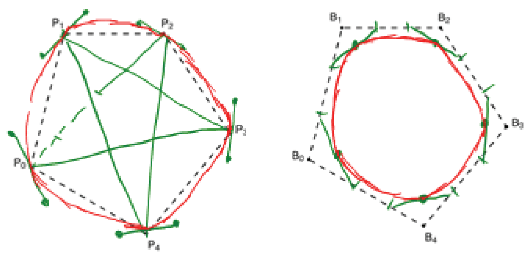
Again, we can force interpolation by repeating the endpoints… *twice*.



Closing the loop

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

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



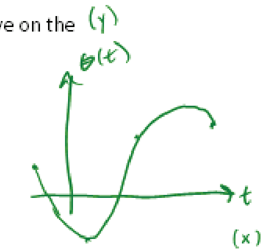
Curves in the animator project

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

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

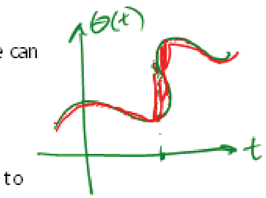
You will actually treat this curve as:

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



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

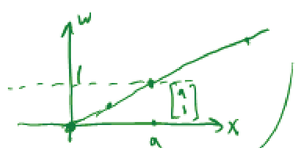
$$\theta(t)$$



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

projective $\quad$ affine

$$\begin{bmatrix} x \\ w \end{bmatrix} \xrightarrow{\div w} \begin{bmatrix} x/w \\ 1 \end{bmatrix}$$



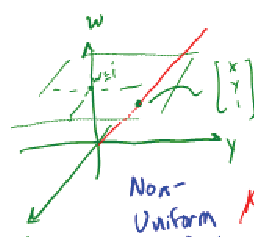
$$\begin{bmatrix} a \\ 1 \end{bmatrix} \text{"="} \begin{bmatrix} 2a \\ 2 \end{bmatrix} \text{"="} \begin{bmatrix} \frac{1}{2}a \\ \frac{1}{2} \end{bmatrix} \text{"="} \begin{bmatrix} sa \\ s \end{bmatrix}$$

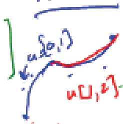
$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{"="} \begin{bmatrix} sx \\ sy \\ s \end{bmatrix}$$

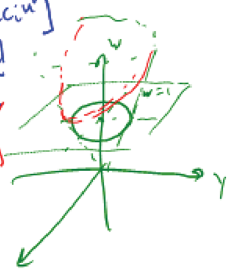
$$Q(u) = \begin{bmatrix} \sum a_i u^i / \sum c_i u^i \\ \sum b_i u^i / \sum c_i u^i \end{bmatrix}$$

rational

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







Non-Uniform Rational B-Splines

NURBS

$$Q(u) = \begin{bmatrix} x(u) \\ y(u) \\ w(u) \end{bmatrix} \rightarrow \begin{bmatrix} x(u)/w(u) \\ y(u)/w(u) \\ 1 \end{bmatrix}$$

$$\left(\frac{2u}{1+u^2}\right)^2 + \left(\frac{1-u^2}{1+u^2}\right)^2 = \frac{4u^2 + 1 - 2u^2 + u^4}{1 + 2u^2 + u^4} = \frac{1 + 2u^2 + u^4}{1 + 2u^2 + u^4} = 1$$