# Parametric Curves



## Reading

- Foley, Section 11.2

**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.

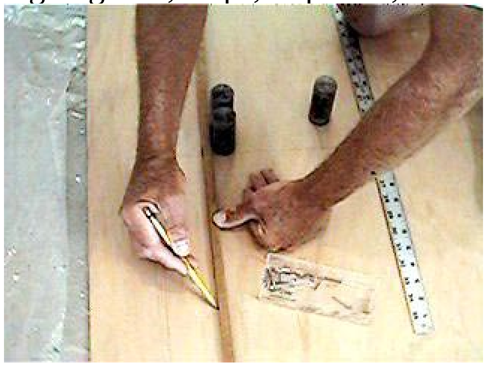


## Curves before computers

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.





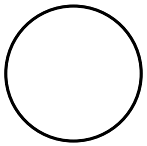
## Motivation for curves

What do we use curves for?

- building models
- movement paths
- animation

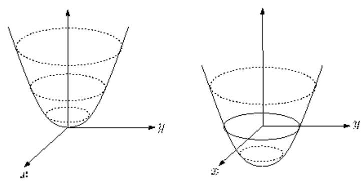
# Mathematical curve representation

- Explicit $y=f(x)$
  - what if the curve isn't a function?



- Implicit $f(x,y) = 0$
  - hard to work with

$$x^2 + y^2 - R^2 = 0$$



- Parametric $(f(u),g(u))$

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

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



# Parametric polynomial curves

We'll use parametric curves 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



# Cubic curves

Fix n=3

For simplicity we define each cubic function within the range

$$0 \le t \le 1$$

$$\mathbf{Q}(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix}$$

$$Q_x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$Q_y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$Q_z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$



# Compact representation

Place all coefficients into a matrix

$$\mathbf{C} = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$$

$$Q(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix} = \mathbf{T} \cdot \mathbf{C}$$

$$\frac{d}{dt} Q(t) = Q'(t) = \frac{d}{dt}\left(\mathbf{T} \cdot \mathbf{C}\right) = \frac{d}{dt}\mathbf{T} \cdot \mathbf{C} + \mathbf{T} \cdot \frac{d}{dt}\mathbf{C} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot \mathbf{C}$$

## Controlling the cubic

**Q**: How many constraints do we need to specify to fully determine the cubic **Q**(t)?



## Constraining the cubics

Redefine **C** as a product of the **basis matrix M** and the 4-element column vector of constraints or **geometry vector G**

$$\mathbf{C} = \mathbf{M} \cdot \mathbf{G}$$

$$\mathbf{Q}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} G_{1x} & G_{1y} & G_{1z} \\ G_{2x} & G_{2y} & G_{2z} \\ G_{3x} & G_{3y} & G_{3z} \\ G_{4x} & G_{4y} & G_{4z} \end{bmatrix}$$

$$= \mathbf{T} \cdot \mathbf{M} \cdot \mathbf{G}$$



## Hermite Curves

Determined by

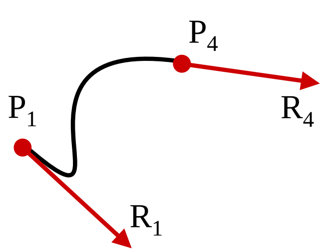
- endpoints $P_1$ and $P_4$
- tangent vectors at the endpoints $R_1$ and $R_4$

So

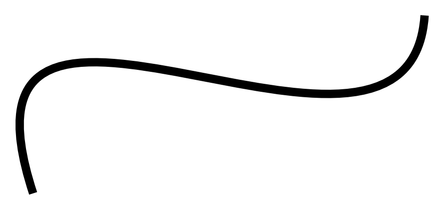
$$\mathbf{Q}(t) = \mathbf{T} \cdot \mathbf{M}_h \cdot \mathbf{G}_h$$

Where

$$\mathbf{G}_h = \begin{bmatrix} P_{1x} & P_{1y} & P_{1z} \\ P_{4x} & P_{4y} & P_{4z} \\ R_{1x} & R_{1y} & R_{1z} \\ R_{4x} & R_{4y} & R_{4z} \end{bmatrix}$$





## Computing Hermite basis matrix

The constraints on Q(0) and Q(1) are found by direct substitution:

$$Q(0) = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{M}_h \cdot \mathbf{G}_h$$

$$Q(1) = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \cdot \mathbf{M}_h \cdot \mathbf{G}_h$$

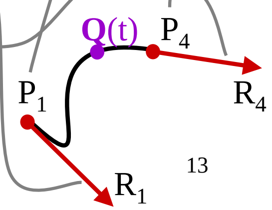
Tangents are defined by

$$Q'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot \mathbf{M}_h \cdot \mathbf{G}_h$$

so constraints on tangents are:

$$Q'(0) = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \cdot \mathbf{M}_h \cdot \mathbf{G}_h$$

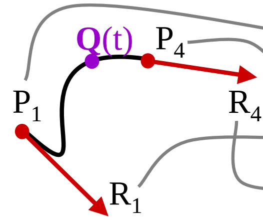
$$Q'(1) = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} \cdot \mathbf{M}_h \cdot \mathbf{G}_h$$






## Computing Hermite basis matrix

Collecting all constraints we get




$$\begin{bmatrix} P_{1x} & P_{1y} & P_{1z} \\ P_{4x} & P_{4y} & P_{4z} \\ R_{1x} & R_{1y} & R_{1z} \\ R_{4x} & R_{4y} & R_{4z} \end{bmatrix} = \mathbf{G}_h = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \mathbf{M}_h \cdot \mathbf{G}_h$$

So

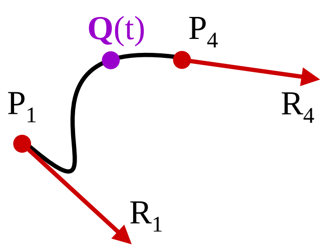
$$\mathbf{M}_h = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$



## Computing a point

Given two endpoints ($P_1$,$P_4$) and two endpoint tangent vectors ($R_1$ , $R_4$):

So




$$\mathbf{Q}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_4 \\ \mathbf{R}_1 \\ \mathbf{R}_4 \end{bmatrix}$$

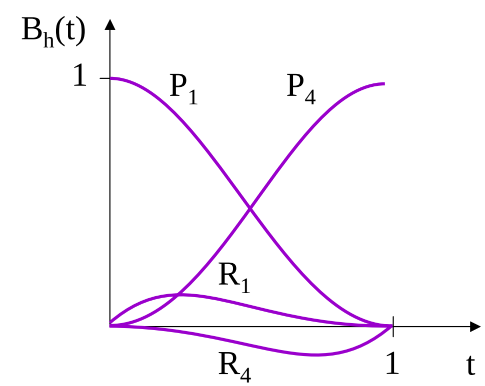


## Blending Functions

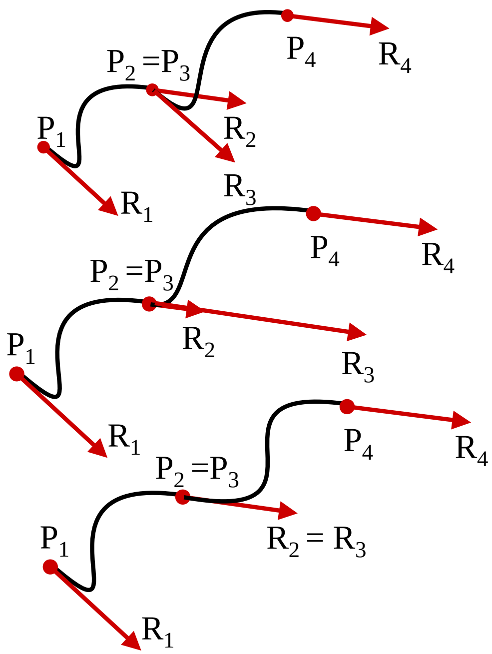
Polynomials weighting each element of the geometry vector

$$\mathbf{Q}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_4 \\ \mathbf{R}_1 \\ \mathbf{R}_4 \end{bmatrix}$$

$$= \mathbf{B}_h(t) \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_4 \\ \mathbf{R}_1 \\ \mathbf{R}_4 \end{bmatrix}$$



## Continuity of Splines



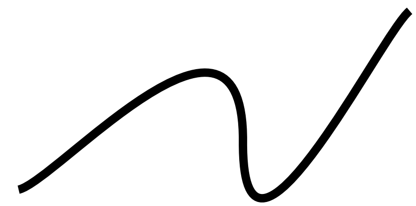

$C^0$: points coincide, velocities don't

$G^1$: points coincide, velocities have same direction

$C^1$: points and velocities coincide

**Q**: What's $C^2$?

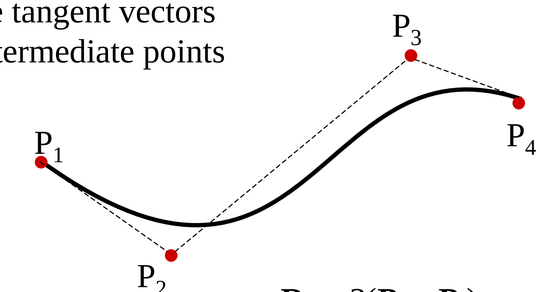






## Bézier Curves

Indirectly specify the tangent vectors by specifying two intermediate points




$$\mathbf{R}_1 = 3(\mathbf{P}_2 - \mathbf{P}_1)$$
$$\mathbf{R}_4 = 3(\mathbf{P}_4 - \mathbf{P}_3)$$

$$\mathbf{G}_b = \begin{bmatrix} P_{1x} & P_{1y} & P_{1z} \\ P_{2x} & P_{2y} & P_{2z} \\ P_{3x} & P_{3y} & P_{3z} \\ P_{4x} & P_{4y} & P_{4z} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{bmatrix}$$



## Bézier basis matrix

Establish the relation between the Hermite and Besier geometry vectors:

$$\mathbf{R}_1 = 3(\mathbf{P}_2 - \mathbf{P}_1)$$
$$\mathbf{R}_4 = 3(\mathbf{P}_4 - \mathbf{P}_3)$$

$$\mathbf{G}_h = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_4 \\ \mathbf{R}_1 \\ \mathbf{R}_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{bmatrix} = \mathbf{M}_{bh}\mathbf{G}_b$$

# Bézier basis matrix

$$\mathbf{Q}(t) = \mathbf{T}\cdot\mathbf{M}_h\cdot\mathbf{G}_h = \mathbf{T}\cdot\mathbf{M}_h\cdot(\mathbf{M}_{hb}\cdot\mathbf{G}_b)$$
$$= \mathbf{T}\cdot(\mathbf{M}_h\cdot\mathbf{M}_{hb})\cdot\mathbf{G}_b = \mathbf{T}\cdot\mathbf{M}_b\cdot\mathbf{G}_b$$

$$\mathbf{M}_b = \mathbf{M}_h\mathbf{M}_{hb} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

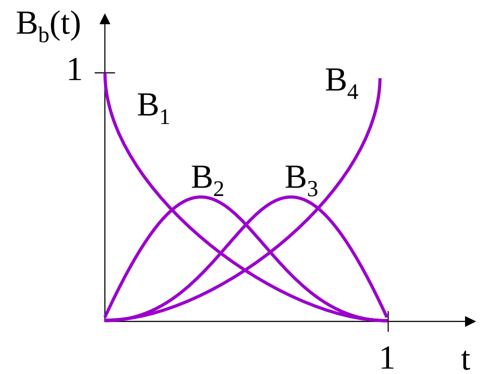
$$\mathbf{Q}(t) = \mathbf{T}\cdot\mathbf{M}_b\cdot\mathbf{G}_b$$



# Bézier Blending Functions

a.k.a. Bernstein polynomials

$$\mathbf{Q}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{bmatrix} = \mathbf{B}_b(t) \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{bmatrix}$$






# Alternative Bézier Formulation

$$Q(t) = \sum_{i=0}^{3} P_i \binom{3}{i} t^i (1-t)^{3-i}$$

$$\mathbf{Q}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{bmatrix}$$

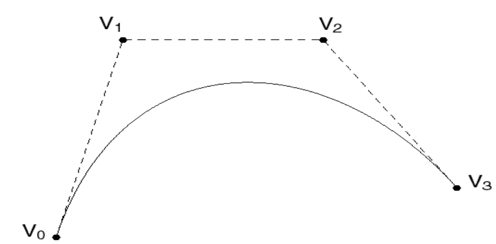
$$Q(t) = \sum_{i=0}^{n} P_i \binom{n}{i} t^i (1-t)^{n-i}$$



# Displaying Bézier curves

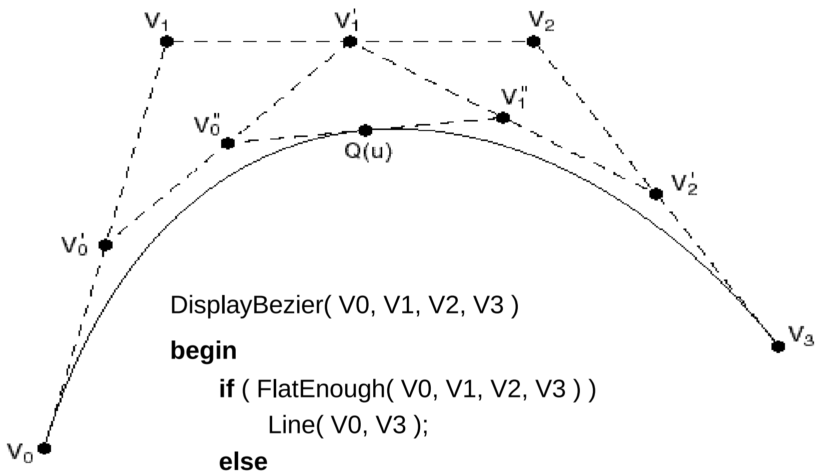
How could we draw one of these things?

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




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

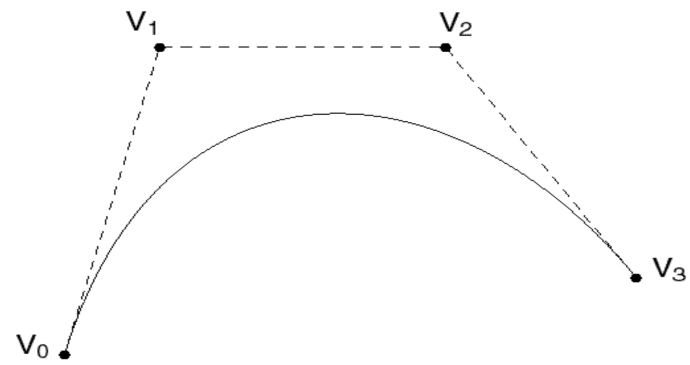
## Subdivide and conquer




```
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:

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



## More complex curves

Suppose we want to draw a more complex curve.

Why not use a high-order Bézier?

Instead, we'll splice together a curve from individual segments that are cubic Béziers.

Why cubic?

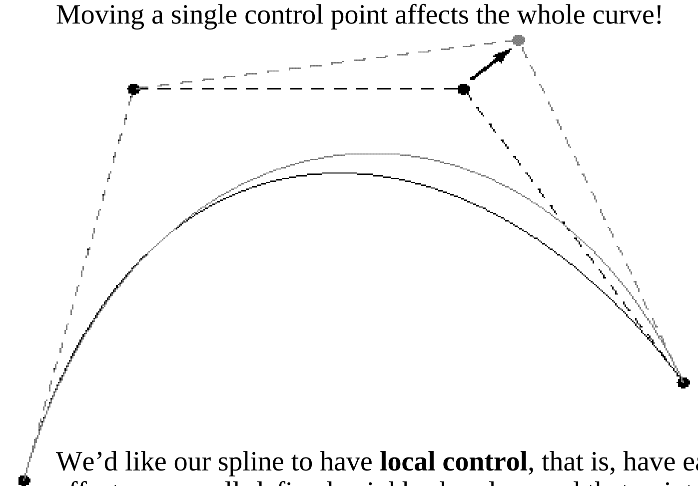
There are three properties we'd like to have in our newly constructed splines…



## Local control

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

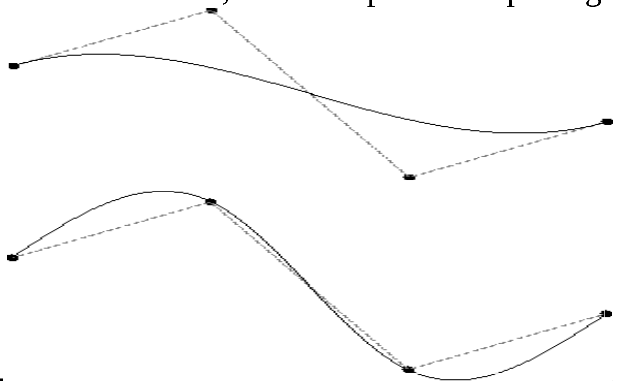
Moving a single control point affects the whole curve!



We'd like our spline 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 spline that is **interpolating**, that is, that always passes through every control point.

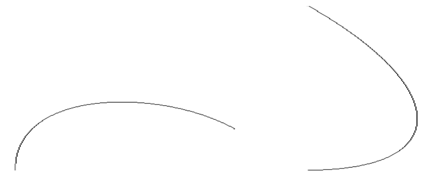


# Continuity

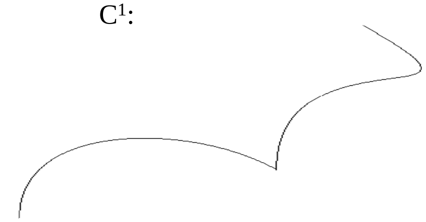
We want our curve to have **continuity**. There shouldn't be an abrupt change when we move from one segment to the next.

There are nested degrees of continuity:

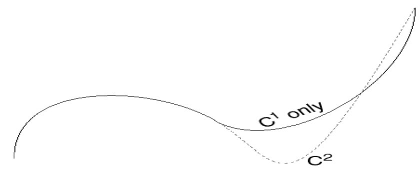
$C^0$:



$C^1$:



$C^2$:






# Ensuring continuity

Let's look at continuity first.

Since the functions defining a Bézier curve are polynomial, all their derivatives exist and are continuous.

Therefore, we only need to worry about the derivatives at the endpoints of the curve.

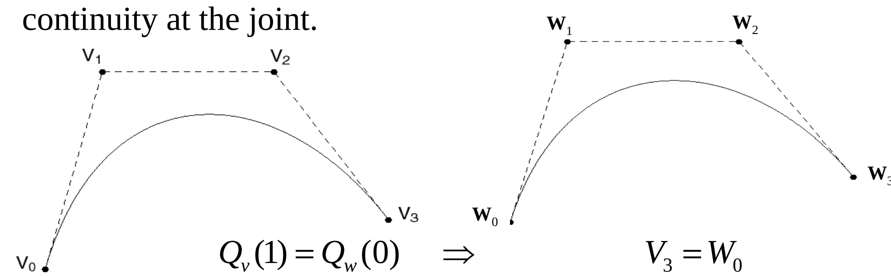
First, we'll rewrite our equation for $Q(t)$ in matrix form:

$$Q(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & \\ -3 & 3 & & \\ 1 & & & \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$



# Ensuring $C^2$ continuity

Suppose we want to join two cubic Bézier curves $(V_0,V_1,V_2,V_3)$ and $(W_0,W_1,W_2,W_3)$ so that there is $C^2$ continuity at the joint.




$$Q_v(1) = Q_w(0) \quad \Rightarrow \quad V_3 = W_0$$

$$Q_v'(1) = Q_w'(0) \quad \Rightarrow \quad V_3 - V_2 = W_1 - W_0$$

$$Q_v''(1) = Q_w''(0) \quad \Rightarrow \quad V_1 - 2V_2 + V_3 = W_0 - 2W_1 + W_2$$

$$\Downarrow$$

$$\boxed{W_2 = V_1 + 4V_3 - 4V_2}$$

$$Q'(0) = 3(V_1 - V_0)$$
$$Q'(1) = 3(V_3 - V_2)$$
$$Q''(0) = 6(V_0 - 2V_1 + V_2)$$
$$Q''(1) = 6(V_1 - 2V_2 + V_3)$$

## A-frames and continuity

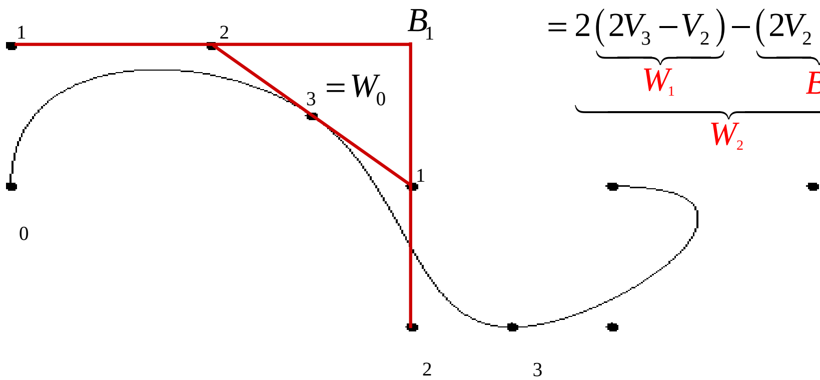
Let’s try to get some geometrical intuition about what this last continuity equation means.

If *a* and *b* are points, what is (2*a*-*b*)?

$$_2 = V_1 + 4V_3 - 4V_2$$

$$= 2\underbrace{\left(2V_3 - V_2\right)}_{W_1} - \underbrace{\left(2V_2 - V_1\right)}_{B_1}$$

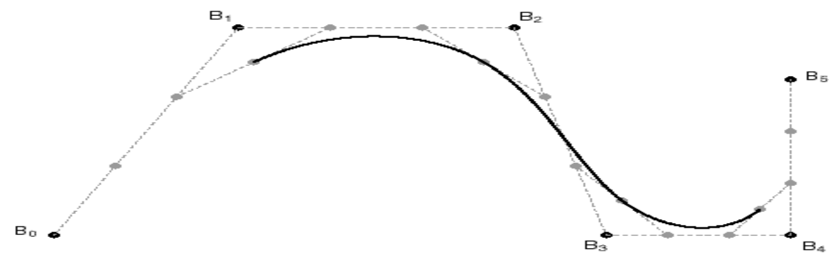
$W_2$






## 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.

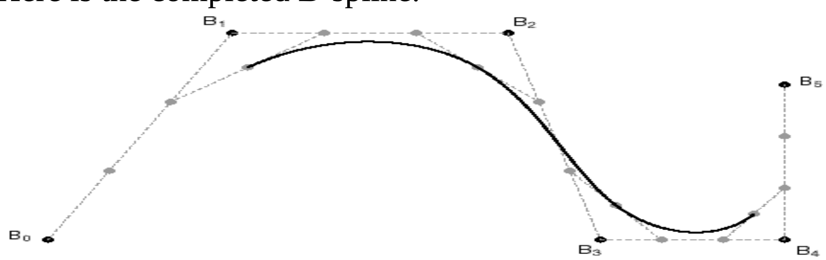



These are called **B-splines**. The starting set of points are called **de Boor points**.



## B-splines

Here is the completed B-spline.




$$_0 = \frac{1}{2}\left(B_0 + \frac{2}{3}(B_1 - B_0) + B_1 + \frac{1}{3}(B_2 - B_1)\right)$$

$$V_1 = B_1 + \frac{1}{3}(B_2 - B_1)$$

$$_2 = B_0 + \frac{2}{3}(B_1 - B_0)$$

$$V_3 = \ldots$$

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

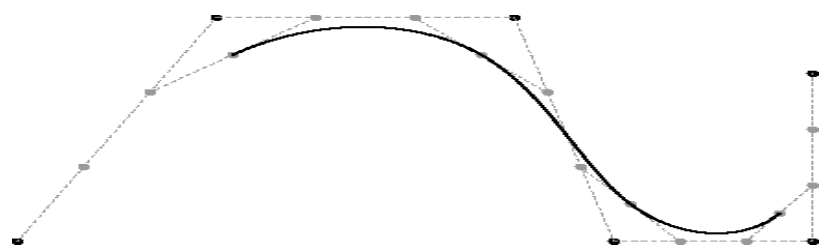


## Endpoints of B-splines

We can see that B-splines don’t interpolate the de Boor points.

It would be nice if we could at least control the *endpoints* of the splines explicitly.

There’s a hack to make the spline begin and end at control points by repeating them.



# B-spline basis matrix

$$\mathbf{Q}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{bmatrix}$$

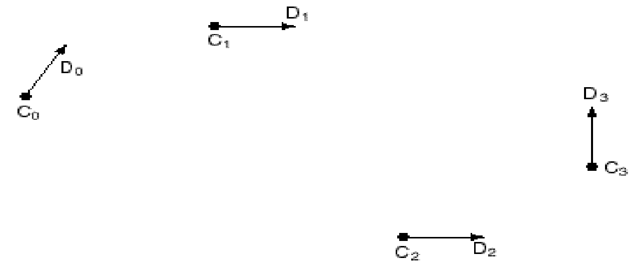


# $C^2$ interpolating splines

Interpolation is a really handy property to have.

How can we keep the $C^2$ continuity we get with B-splines but get interpolation, too?

Here's the idea behind **$C^2$ interpolating splines**. Suppose we had cubic Béziers connecting our control points $C_0$, $C_1$, $C_2$, …, and that we somehow knew the first derivative of the spline at each point.

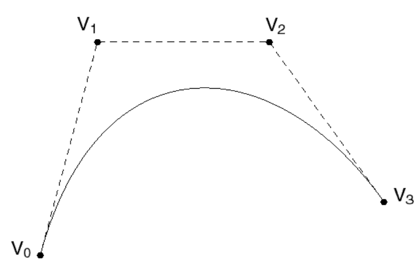



What are the $V$ and $W$ control points in terms of $C$s and $D$s?



# Derivatives at the endpoints

$$Q'(0) = 3(V_1 - V_0)$$
$$Q'(1\ ) = 3(V_3 - V_2)$$
$$Q''(0) = 6(V_0 - 2V_1 + V_2)$$
$$Q''(1\ ) = 6(V_1 - 2V_2 + V_3)$$




$$Q''(t) = \begin{bmatrix} 6t & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & \\ -3 & 3 & & \\ 1 & & & \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{bmatrix}$$

In general, the $n$th derivative at an endpoint depends only on the $n$+1 points nearest that endpoint.



# Finding the derivatives

Now what we need to do is solve for the derivatives. To do this we'll use the $C^2$ continuity requirement.

$$\begin{aligned} {}_0 &= C_0 & {}_0 &= C_1 \\ V_1 &= C_0 + \tfrac{1}{3}D_0 & W_1 &= C_1 + \tfrac{1}{3}D_1 \\ {}_2 &= C_1 - \tfrac{1}{3}D_1 & {}_2 &= C_2 - \tfrac{1}{3}D_2 \\ V_3 &= C_1 & W_3 &= C_2 \end{aligned}$$

$$6(V_1 - 2V_2 + V_3) = 6(W_0 - 2W_1 + W_2)$$
$$\Downarrow$$
$$D_0 + 4D_1 + D_2 = 3(C_2 - C_0)$$

## Finding the derivatives, cont.

Here's what we've got so far:

$$\begin{aligned} D_0 + 4D_1 + D_2 &= 3(C_2 - C_0) \\ D_1 + 4D_2 + D_3 &= 3(C_3 - C_1) \\ &\vdots \\ D_{m-2} + 4D_{m-1} + D_m &= 3(C_m - C_{m-2}) \end{aligned}$$

How many equations is this?

How many unknowns are we solving for?



## Not quite done yet

We have two additional degrees of freedom, which we can nail down by imposing more conditions on the curve.

There are various ways to do this. We'll use the variant called **natural C² interpolating splines**, which requires the second derivative to be zero at the endpoints.

This condition gives us the two additional equations we need. At the $C_0$ endpoint, it is:

$$6(V_0 - 2V_1 + V_2) = 0$$



## Solving for the derivatives

Let's collect our *m*+1 equations into a single linear system:

$$\begin{bmatrix} 2 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \vdots \\ D_{m-1} \\ D_m \end{bmatrix} = \begin{bmatrix} 3(C_1 - C_0) \\ 3(C_2 - C_0) \\ 3(C_3 - C_1) \\ \vdots \\ 3(C_m - C_{m-2}) \\ 3(C_m - C_{m-1}) \end{bmatrix}$$

It's easier to solve than it looks.

We can use **forward elimination** to zero out everything below the diagonal, then **back substitution** to compute each *D* value.



## Forward elimination

First, we eliminate the elements below the diagonal:

$$\begin{bmatrix} 2 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} \mathbf{D}_0 \\ \mathbf{D}_1 \\ \mathbf{D}_2 \\ \vdots \\ \mathbf{D}_{m-1} \\ \mathbf{D}_m \end{bmatrix} = \begin{bmatrix} \mathbf{E}_0 \\ \mathbf{E}_1 \\ \mathbf{E}_2 \\ \vdots \\ \mathbf{E}_{m-1} \\ \mathbf{E}_m \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & & & & \\ 0 & 7/2 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} \mathbf{D}_0 \\ \mathbf{D}_1 \\ \mathbf{D}_2 \\ \vdots \\ \mathbf{D}_{m-1} \\ \mathbf{D}_m \end{bmatrix} = \begin{bmatrix} \mathbf{F}_0 = \mathbf{E}_0 \\ \mathbf{F}_1 = \mathbf{E}_1 - (1/2)\mathbf{E}_0 \\ \mathbf{E}_2 \\ \vdots \\ \mathbf{E}_{m-1} \\ \mathbf{E}_m \end{bmatrix}$$

## Back subsitution

The resulting matrix is **upper diagonal**:

**UD = F**

$$\begin{bmatrix} u_{11} & \cdots & u_{1m} \\ & & \\ & & \\ & \ddots & \vdots \\ & & \\ & & u_{mm} \end{bmatrix} \begin{bmatrix} \mathbf{D}_0 \\ \mathbf{D}_1 \\ \mathbf{D}_2 \\ \vdots \\ \mathbf{D}_{m-1} \\ \mathbf{D}_m \end{bmatrix} = \begin{bmatrix} \mathbf{F}_0 \\ \mathbf{F}_1 \\ \mathbf{F}_2 \\ \vdots \\ \mathbf{F}_{m-1} \\ \mathbf{F}_m \end{bmatrix}$$

We can now solve for the unknowns by back substitution:

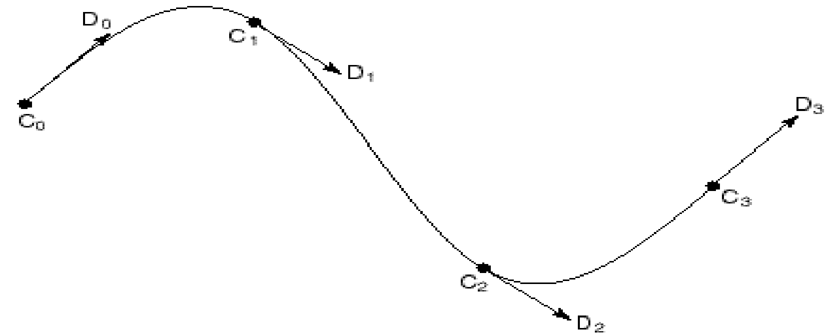
$$u_{mm}\mathbf{D}_m = \mathbf{F}_m$$

$$u_{m-1m-1}\mathbf{D}_{m-1} + u_{m-1m}\mathbf{D}_m = \mathbf{F}_{m-1}$$



## $C^2$ interpolating spline

Once we've solved for the real $D_i$s, we can plug them in to find our Bézier control points and draw the final spline:




Have we lost anything?

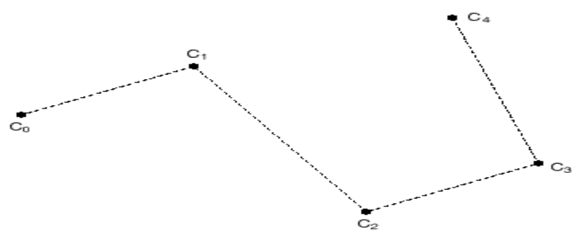


## A third option

If we're willing to sacrifice $C^2$ continuity, we can get interpolation *and* local control.

Instead of finding the derivatives by solving a system of continuity equations, we'll just pick something arbitrary but local.

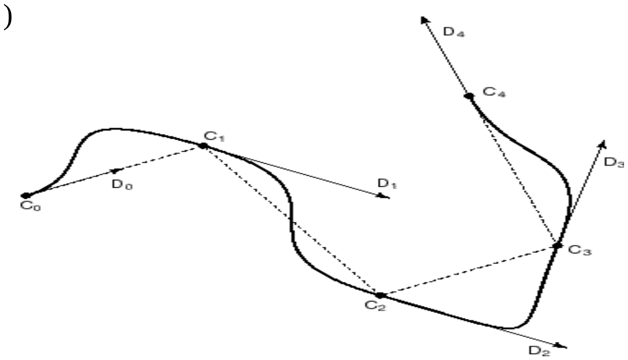
If we set each derivative to be a constant multiple of the vector between the previous and next controls, we get a **Catmull-Rom spline**.




## Catmull-Rom splines

The math for Catmull-Rom splines is pretty simple:

$$D_0 = C_1 - C_0$$
$$D_1 = \tfrac{1}{2}(C_2 - C_0)$$
$$D_2 = \tfrac{1}{2}(C_3 - C_1)$$
$$\vdots$$
$$D_n = C_n - C_{n-1}$$



## Catmull-Rom basis matrix

$$\mathbf{Q}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{bmatrix}$$



## Summary

- Enforcing constraints on cubic functions
- The meaning of basis matrix and geometry vector
- General procedure for computing the basis matrix
- Properties of Hermite and Bézier splines
- The meaning of blending functions
- Enforcing continuity across multiple curve segments
- How to display Bézier curves with line segments.
- Meanings of $C^k$ continuities.
- Geometric conditions for continuity of cubic splines.
- Properties of $C^2$ interpolating splines, B-splines, and Catmull-Rom splines.