Interpolating curves

Reading

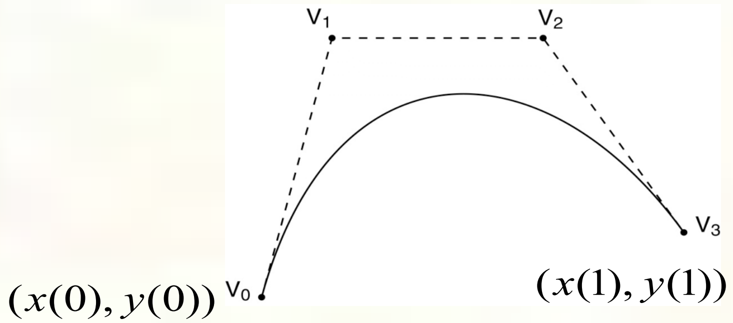
- Optional
  - Bartels, Beatty, and Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*, 1987. (See course reader.)

Parametric curve review

Parametric curves

- We use parametric curves, *Q(u)=(x(u),y(u))*, where *x(u)* and *y(u)* are cubic polynomials:

$$x(u) = Au^3 + Bu^2 + Cu + D$$
$$y(u) = Eu^3 + Fu^2 + Gu + H$$



- Advantages:
  - easy (and efficient) to compute
  - "well behaved"
  - infinitely differentiable
- We also assume that *u* varies from 0 to 1

Various ways to set A,B,C,D

$$x(u) = Au^3 + Bu^2 + Cu + D$$

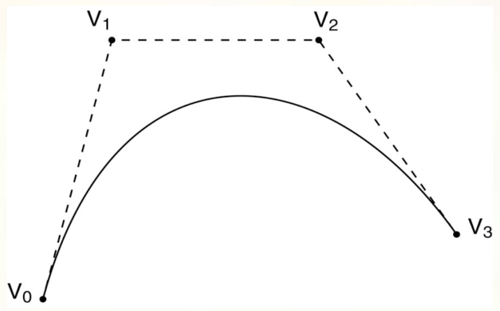
0) Directly – non-intuitive; not very useful.

1) Set positions and derivatives of endpoints:
"Hermite Curve"

2) Use "control points" that indirectly influence the curve:

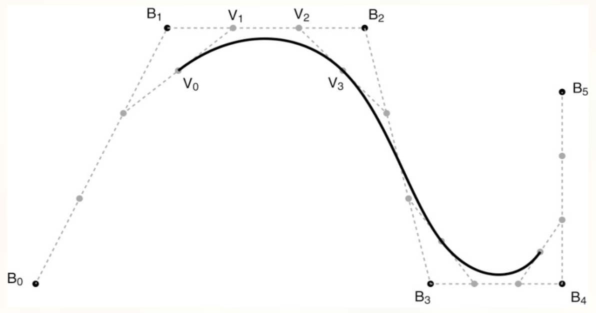
"Bezier curve":

- interpolates endpoints
- does not interpolate middle control points

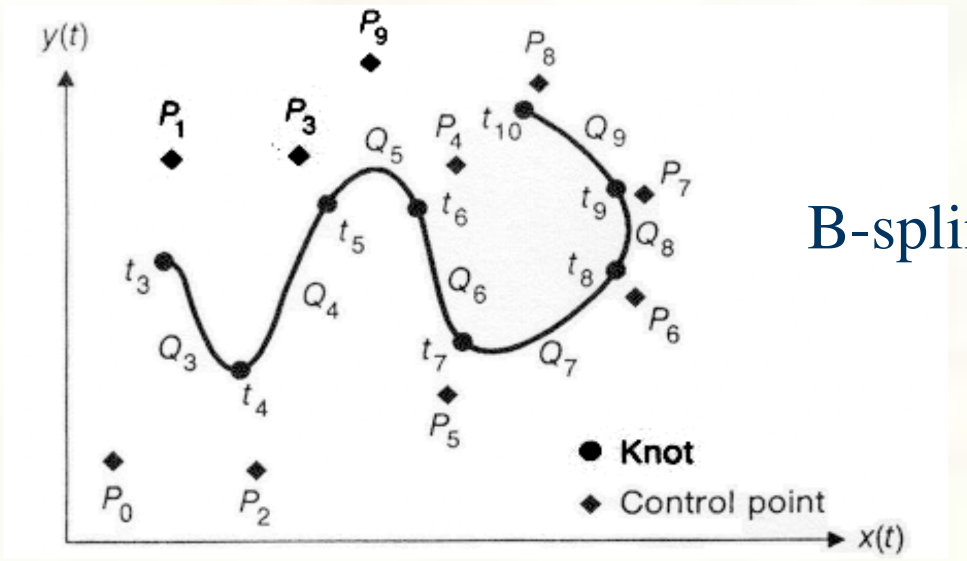


"B-spline"

- does not interpolate ANY control points



Splines = join cubic curves



B-spline

<u>Considerations</u>

What kind of continuity at join points ("knots")?

C0 = value
C1 = first derivative
C2 = second derivative

How do control points work?

Spline summary

- Joined Hermite curves:
  - C1 continuity
  - Interpolates control points
- B-splines:
  - C2 continuity
  - Does not interpolate control points

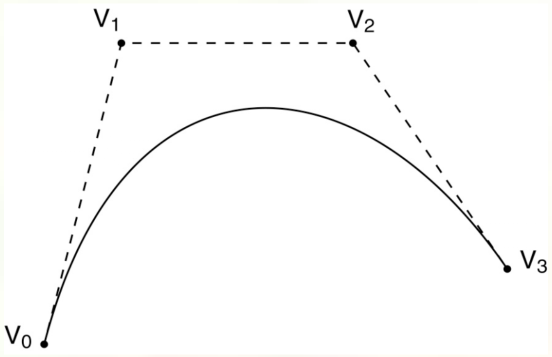
Can we get…
- C2 continuity
- Interpolates control points

That's what we'll talk about towards the end of this lecture.
But first, some other useful tips.

Useful tips for Bézier curves

Displaying Bézier curves

- How could we draw one of these things?

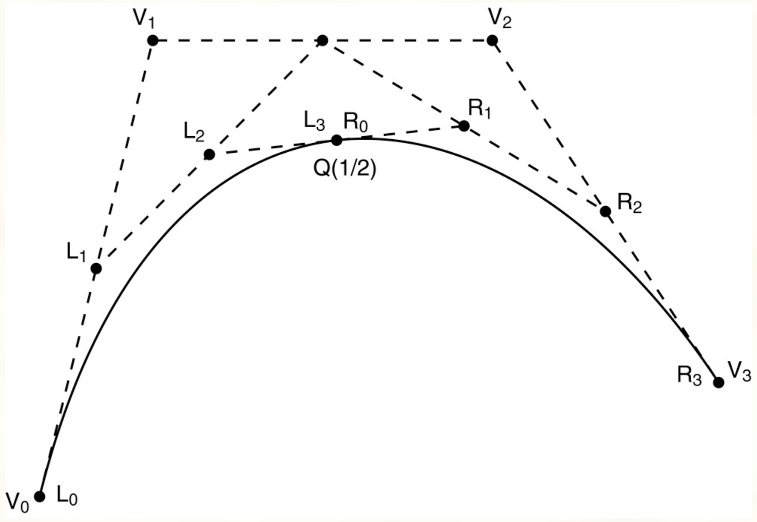


- It would be nice if we had an *adaptive* algorithm, that would take into account flatness.

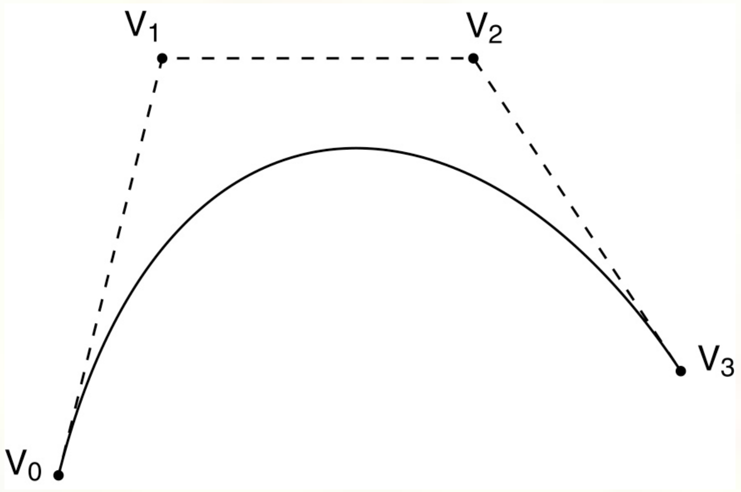
```
DisplayBezier( V0, V1, V2, V3 )
begin
    if ( FlatEnough( V0, V1, V2, V3 ) )
        Line( V0, V3 );
    else
        something;
end;
```

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:

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

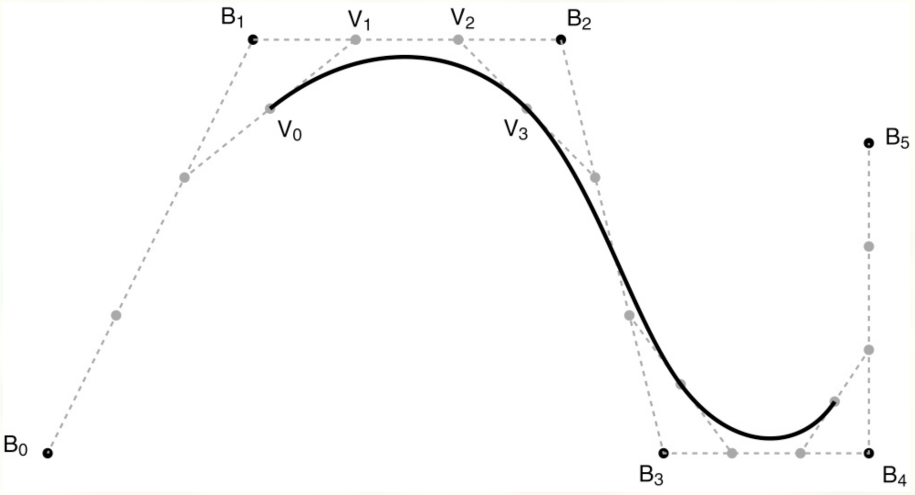
Tips for B-splines

B-spline:

- C2 continuity
- does not interpolate any ctrl points

Endpoints of B-splines

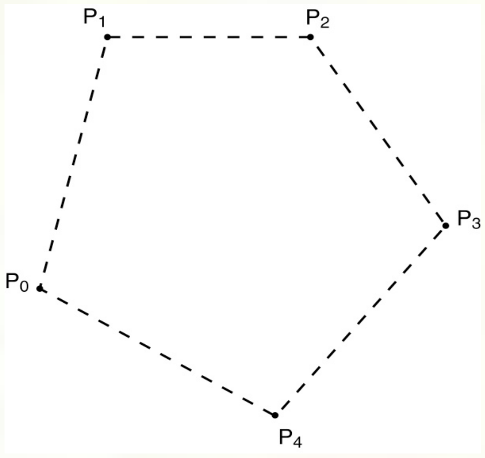
- We can see that B-splines don’t interpolate the control points.
- It would be nice if we could at least control the *endpoints* of the splines explicitly.
- There’s a trick to make the spline begin and end at control points by repeating them.
- In the example below, let’s force interpolation of the last endpoint: (use endpoint 3 times)

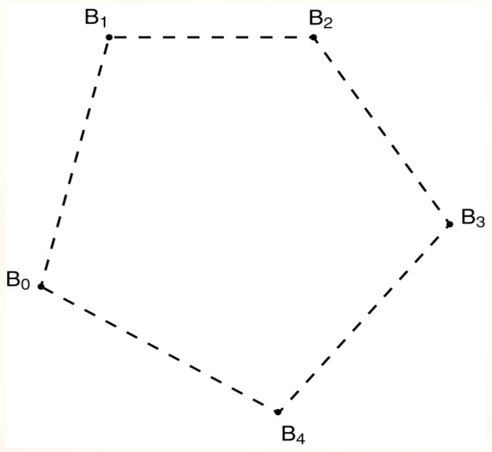


Tips for animator project

Closing the loop

- What if we want a closed curve, i.e., a loop?
- With Catmull-Rom and B-spline curves, this is easy:

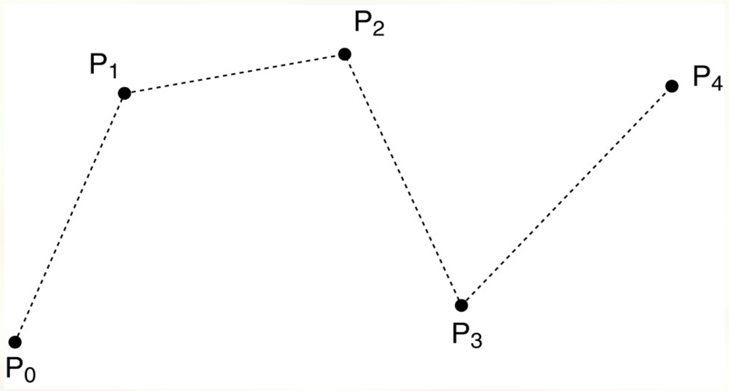




C2 interpolating curves

Simple interpolating splines

- Join several Hermite curves:
  - Make derivatives match
  - You still have ability to pick what that matched derivative is.

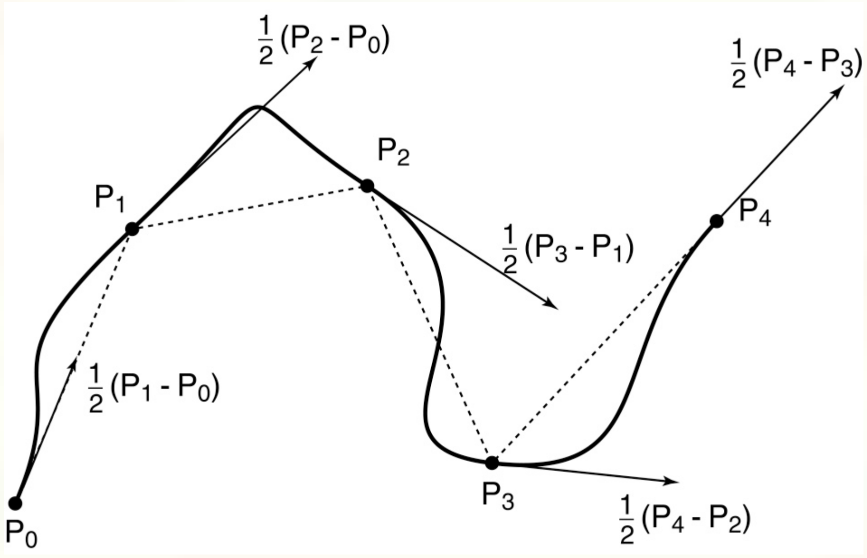


Catmull-Rom splines

- 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:

$$\mathbf{p}_i^u = \frac{\mathbf{p}_{i+1} - \mathbf{p}_{i-1}}{2}$$

$$\mathbf{p}_{i+1}^u = \frac{\mathbf{p}_{i+2} - \mathbf{p}_i}{2}$$



for any two consecutive interior points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ (we can deal with the endpoints separately if need be)

Catmull-Rom splines

$$\mathbf{p}(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_i^u \\ \mathbf{p}_{i+1}^u \end{bmatrix}$$

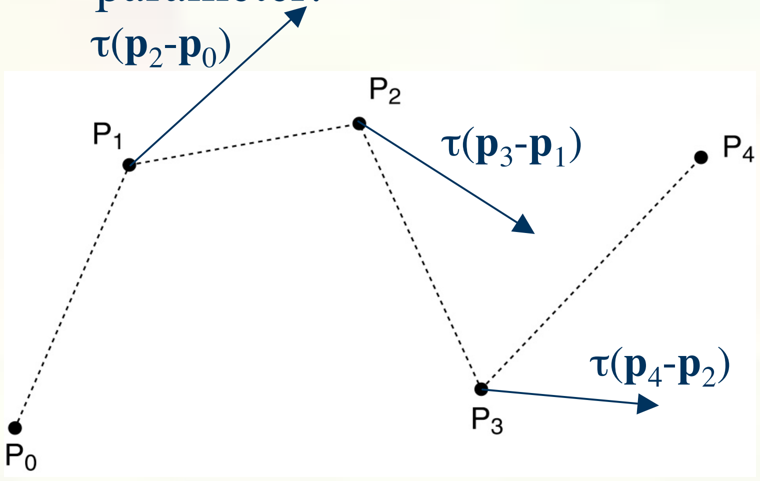
$$= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ (\mathbf{p}_{i+1} - \mathbf{p}_{i-1})/2 \\ (\mathbf{p}_{i+2} - \mathbf{p}_i)/2 \end{bmatrix}$$

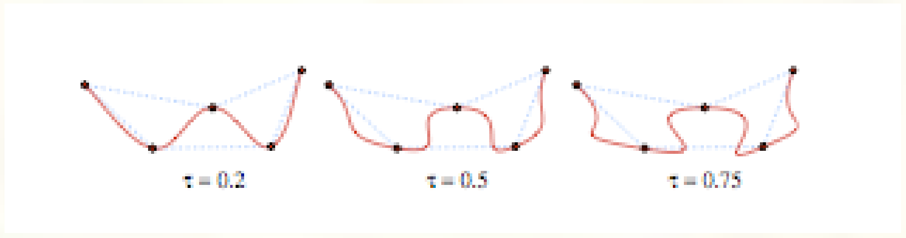
$$= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1/2 & 0 & 1/2 & 0 \\ 0 & -1/2 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix}$$

$$= \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix}$$

Cardinal splines -tension control

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



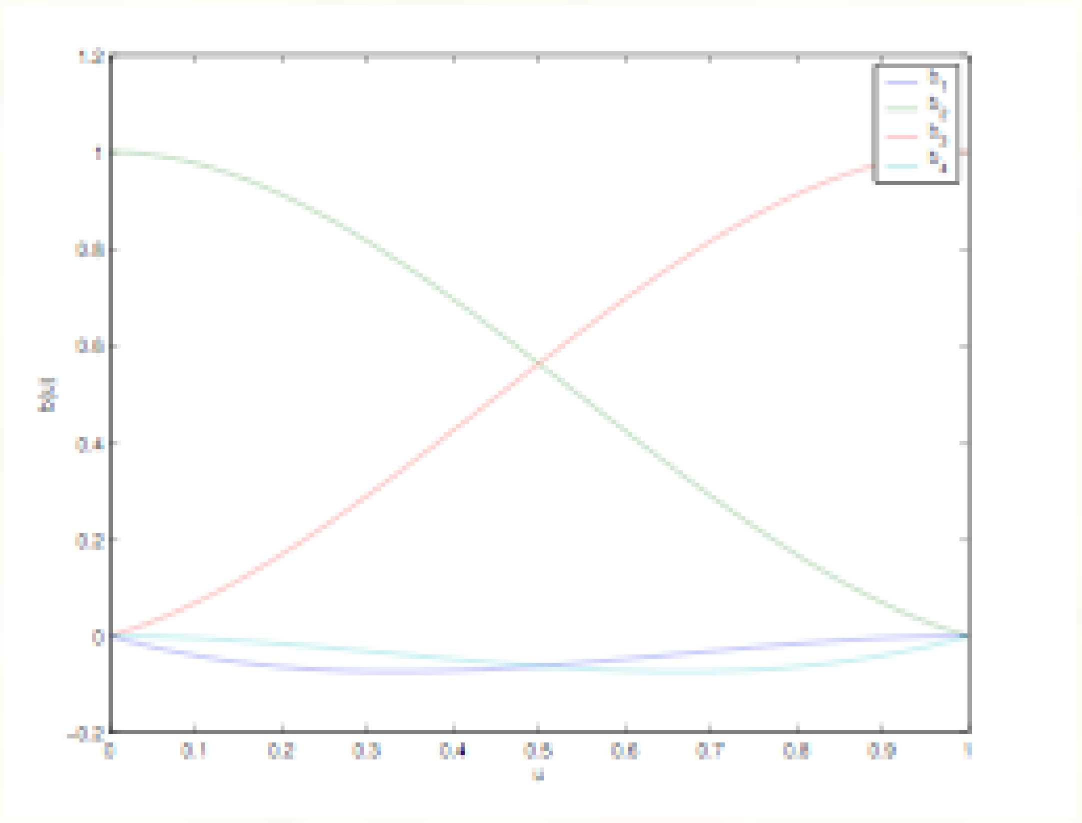


- The parameter τ controls the tension. Catmull-Rom uses $\tau = 1/2$.

$$\mathbf{p}_i^u = \tau(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$$
$$\mathbf{p}_{i+1}^u = \tau(\mathbf{p}_{i+2} - \mathbf{p}_i)$$

$$\mathbf{p}(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \tau \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix}$$

Catmull-Rom blending functions



C^2 interpolating splines

- How can we keep the C^2 continuity we get with B-splines but get interpolation, too?
- Again start with connected cubic curves.
- Each cubic segment is an Hermite curve for which we get to set the position and derivative of the endpoints.
- That leaves us with a spline that's C^0 and C^1 such as a Catmull-Rom or Cardinal spline.
- But interestingly, there are other ways to choose the values of the (shared) first derivatives at the join points.
- Is there a way to set those derivatives to get other useful properties?

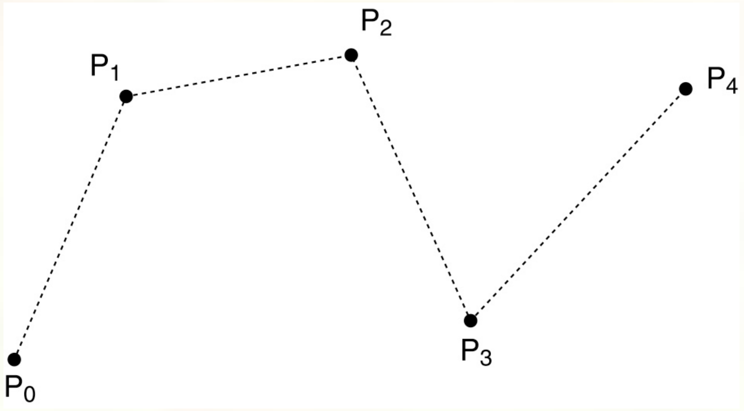
Find second derivatives

So far, we have:

C^0, C^1 continuity

Derivatives are still free, as ‘$D_0 \ldots D_4$’

Compute second derivatives at both sides of every join point:

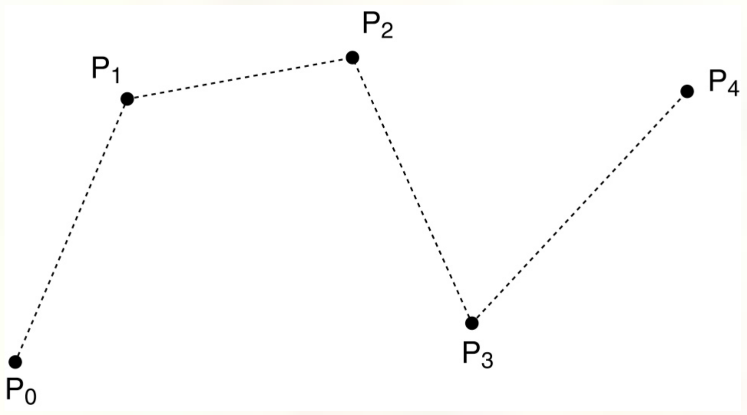


For $\mathbf{p}_1$: $Q_0''(1) = 6\mathbf{p}_0 - 6\mathbf{p}_1 + 2\mathbf{D}_0 + 4\mathbf{D}_1 \quad Q_1''(0) = -6\mathbf{p}_1 + 6\mathbf{p}_2 - 4\mathbf{D}_1 - 2\mathbf{D}_2$

For $\mathbf{p}_2$: $Q_1''(1) = 6\mathbf{p}_1 - 6\mathbf{p}_2 + 2\mathbf{D}_1 + 4\mathbf{D}_2 \quad Q_2''(0) = -6\mathbf{p}_2 + 6\mathbf{p}_3 - 4\mathbf{D}_2 - 2\mathbf{D}_3$

…

Match the second derivatives



Now, symbolically set the second derivatives to be equal.

For $\mathbf{p}_1$

$$6\mathbf{p}_0 - 6\mathbf{p}_1 + 2\mathbf{D}_0 + 4\mathbf{D}_1 = -6\mathbf{p}_1 + 6\mathbf{p}_2 - 4\mathbf{D}_1 - 2\mathbf{D}_2$$
$$3(\mathbf{p}_2 - \mathbf{p}_0) = \mathbf{D}_0 + 4\mathbf{D}_1 + \mathbf{D}_2$$

For $\mathbf{p}_2$

$$6\mathbf{p}_1 - 6\mathbf{p}_2 + 2\mathbf{D}_1 + 4\mathbf{D}_2 = -6\mathbf{p}_2 + 6\mathbf{p}_3 - 4\mathbf{D}_2 - 2\mathbf{D}_3$$
$$3(\mathbf{p}_3 - \mathbf{p}_1) = \mathbf{D}_1 + 4\mathbf{D}_2 + \mathbf{D}_3$$

…

Not quite done yet

- How many equations is this? m-1
- How many unknowns are we solving for? m+1
- 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 C^2 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 P_0 endpoint, it is: $Q_0''(0) = 0$
  - At the P_m endpoint, we have: $Q_{m-1}''(1) = 0$

Solving for the derivatives

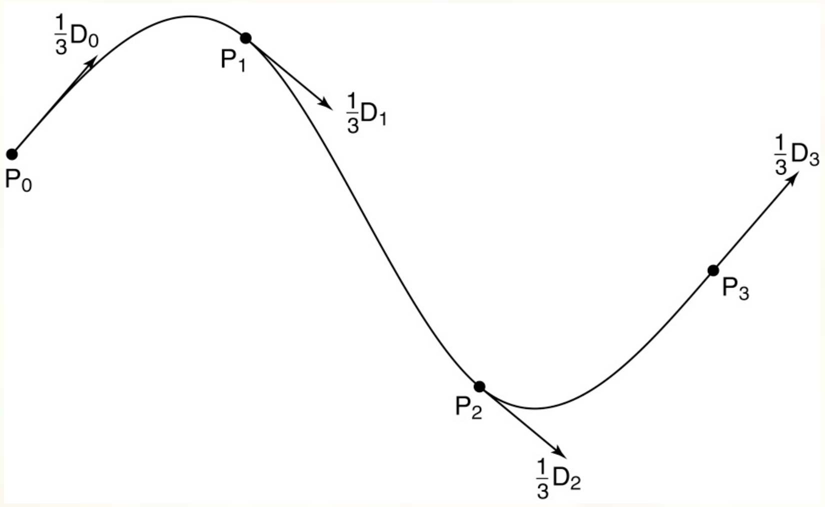
- Let's collect our m+1 equations into a single linear system:

$$\begin{bmatrix} 2 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} \mathbf{D}_0^T \\ \mathbf{D}_1^T \\ \mathbf{D}_2^T \\ \vdots \\ \mathbf{D}_{m-1}^T \\ \mathbf{D}_m^T \end{bmatrix} = \begin{bmatrix} 3(\mathbf{p}_1 - \mathbf{p}_0)^T \\ 3(\mathbf{p}_2 - \mathbf{p}_0)^T \\ 3(\mathbf{p}_3 - \mathbf{p}_1)^T \\ \vdots \\ 3(\mathbf{p}_m - \mathbf{p}_{m-2})^T \\ 3(\mathbf{p}_m - \mathbf{p}_{m-1})^T \end{bmatrix}$$

- It's easier to solve than it looks.
- See the notes from Bartels, Beatty, and Barsky for details.

C^2 interpolating spline

Once we've solved for the real $\mathbf{D}_i$s, we can plug them in to find our Bézier or Hermite curves and draw the final spline:



Have we lost anything?

=> Yes, local control.

Next time: Subdivision curves

- Basic idea:

  Represent a curve as an iterative algorithm, rather than as an explicit function.

- Reading:

  • Stollnitz, DeRose, and Salesin. Wavelets for Computer Graphics: Theory and Applications, 1996, section 6.1-6.3, A.5.
  [Course reader pp. 248-259 and pp. 273-274]