13. Curves

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.

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, e.g., a circle?

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

- Parametric \((x(u), y(u))\)
  - For the circle:
    \[
    x(u) = \cos 2\pi u \\
    y(u) = \sin 2\pi u
    \]

Parametric polynomial curves

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

\[
\begin{align*}
  x(u) &= \sum_{k=0}^{n} a_k u^k \\
  y(u) &= \sum_{k=0}^{n} b_k u^k
\end{align*}
\]

Advantages:

- easy (and efficient) to compute
- infinitely differentiable

We’ll also assume that \( u \) varies from 0 to 1.

de Casteljau’s algorithm

Recursive interpolation:

What if \( u = 0 \)?
What if \( u = 1 \)?

de Casteljau’s algorithm, cont’d

Recursive notation:

What is the equation for \( V_0^1 \)?
In general, 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?

\[ Q(u) = (1-u)V + uV \]

\[ Q(u) = (1-u)V_0^i + uV_1^i \]

\[ Q(u) = (1-u)V + uV \]

\[ Q(u) = (1-u)V_0^i + uV_1^i \]

\[ Q(u) = (1-u)((1-u)V + uV) + u((1-u)V + uV) + \ldots \]

\[ Q(u) = (1-u)^3V_0 + 3u(1-u)^2V_1 + 3u^2(1-u)V_2 + u^3V_3 \]

In general,

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

This defines a class of curves called 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.

```plaintext
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:

\[ \frac{|V_0 - V_1| + |V_1 - V_2| + |V_2 - V_3|}{|V_0 - V_3|} < 1 + \epsilon \]

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!

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.

We’d like our spline to have local control, that is, have each control point affect some well-defined neighborhood around that 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:

$C^{-1}$: $C^0$: 

$C^1$, $C^2$: $C^3$, $C^4$, …:

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 $Q(u)$ in matrix form:

$$Q(u) = [u^3 \quad u^2 \quad u \quad 1] \cdots \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_0^T \\ V_1^T \\ V_2^T \\ V_3^T \end{bmatrix}$$

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.

$C^0 : Q'_V(1) = Q'_W(0)$

$C^1 : Q''_V(1) = Q''_W(0)$

$C^2 : Q'''_V(1) = Q'''_W(0)$

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

Derivatives at the endpoints

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

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

$$V_0 = \ldots \left[ B_0 + \ldots B_1 \right]$$
$$+ \ldots \left[ B_1 + \ldots B_2 \right]$$
$$= \ldots B_0 + \ldots B_1 + \ldots B_2$$
$$V_1 = \ldots B_1 + \ldots B_2$$
$$V_2 = \ldots B_1 + \ldots B_2$$
$$V_3 = \ldots B_1 + \ldots B_2 + \ldots B_3$$

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 trick to make the spline begin and end at control points by repeating them.

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 $C_0, C_1, C_2, \ldots, C_m$, 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 $Cs$ and $Ds$?
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.

$$
\begin{align*}
V_0 &= C_0 & W_0 &= C_1 \\
V_1 &= C_0 + \frac{1}{2}D_0 & W_1 &= C_1 + \frac{1}{2}D_1 \\
V_2 &= C_1 - \frac{1}{2}D_1 & W_2 &= C_2 - \frac{1}{2}D_2 \\
V_3 &= C_1 & W_3 &= C_2
\end{align*}
$$

$$
6(V_1 - 2V_2 + V_3) = 6(W_0 - 2W_1 + W_2)
$$

Finding the derivatives, cont.

Here’s what we’ve got so far:

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

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 $C^2$ 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 $C_0$ endpoint, it is:

$$
6(V_0 - 2V_2 + V_3) = 0
$$

Solving for the derivatives

Let’s collect our $m+1$ equations into a single linear system:

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

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:

\[
\begin{bmatrix}
2 & 1 \\
1 & 4 & 1 \\
1 & 4 & 1 \\
1 & 4 & 1 \\
1 & 4 & 1 \\
1 & 4 & 1 \\
\end{bmatrix}
\begin{bmatrix}
D_0 \\
D_1 \\
D_2 \\
D_3 \\
D_4 \\
D_5 \\
\end{bmatrix}
\begin{bmatrix}
E_0 \\
E_1 \\
E_2 \\
E_3 \\
E_4 \\
E_5 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 1 & 0 & 7 & 2 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
\end{bmatrix}
\begin{bmatrix}
D_0 \\
D_1 \\
D_2 \\
D_3 \\
D_4 \\
D_5 \\
\end{bmatrix}
\begin{bmatrix}
F_0 \\
E_1 \\
E_2 \\
E_3 \\
E_4 \\
E_5 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 1 & 0 & 7 & 2 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
1 & 4 & 1 & 1 & 4 & 1 \\
\end{bmatrix}
\begin{bmatrix}
F_0 \\
E_1 \\
E_2 \\
E_3 \\
E_4 \\
E_5 \\
\end{bmatrix}
\begin{bmatrix}
F_0 \\
F_1 \\
F_2 \\
F_3 \\
F_4 \\
F_5 \\
\end{bmatrix}
\]

The resulting matrix is upper diagonal:

\[
UD = F
\]

\[
\begin{bmatrix}
u_1 & \ldots & u_m
\end{bmatrix}
\begin{bmatrix}
D_0 \\
D_1 \\
D_2 \\
\vdots \\
D_{m-1} \\
D_m
\end{bmatrix}
\begin{bmatrix}
F_0 \\
F_1 \\
F_2 \\
\vdots \\
F_{m-1} \\
F_m
\end{bmatrix}
\]

We can now solve for the unknowns by back substitution:

\[
u_{mm}D_m = F_m
\]

\[
u_{m-1,m}D_{m-1} + u_{m-1,m}D_m = F_{m-1}
\]

Back substitution

C^2 interpolating spline

Once we’ve solved for the real \( D_s \), we can plug them in to find our Bézier control points and draw the final spline:

Have we lost anything?

\[
\begin{align*}
C_0 & \quad \frac{1}{3}D_0 \\
C_1 & \quad \frac{1}{3}D_1 \\
C_2 & \quad \frac{1}{3}D_2 \\
C_3 & \quad \frac{1}{3}D_3 \\
\end{align*}
\]

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.

\[
\begin{align*}
C_0 & \quad \frac{1}{3}D_0 \\
C_1 & \quad \frac{1}{3}D_1 \\
C_2 & \quad \frac{1}{3}D_2 \\
C_3 & \quad \frac{1}{3}D_3 \\
C_4 & \quad \frac{1}{3}D_4
\end{align*}
\]
**Catmull-Rom splines**

The math for Catmull-Rom splines is pretty simple:

\[ V_0 = C_0 \]
\[ V_1 = C_1 + \frac{3}{2} (C_2 - C_0) \]
\[ V_2 = C_2 - \frac{3}{2} (C_3 - C_1) \]
\[ V_3 = C_3 \]

The parameter \( t \) controls the tension.

**Closing the loop**

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

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

![Diagram of closed loop with control points](image)

**Closing the loop, cont’d**

With \( C^2 \) interpolating splines, we have to modify the matrix:

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

We can use a modified forward elimination to zero out everything below the diagonal, then back substitution to compute each \( D \) value.

**Curves in the animator project**

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

\[ Q(u) = (x(u), y(u)) \]

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:

\[ \theta(t) \]

You have to apply some constraints to make sure that \( \theta(t) \) actually is a function.
“Wrapping”

One of the requirements is to implement “wrapping” so that the animation restarts smoothly when looping back to the beginning.

This is a lot like making a closed curve: the calculations for the \( \theta \)-coordinate are exactly the same.

The \( t \)-coordinate is a little trickier: you need to create “phantom” \( t \)-coordinates before and after the first and last coordinates.

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 \( C^2 \) interpolating splines, B-splines, and Catmull-Rom splines.
- Geometric and algebraic construction of B-splines and Catmull-Rom splines.
- How to construct closed loop splines.