CMSC427 Notes on piecewise parametric curves: Hermite, Catmull-Rom, and Bezier

I. Parametric curves and surfaces

Model shapes and behavior with parametric curves
Have done lines, circles, cylinders, superellipses, and others
But limitations – how can we model an arbitrary shape? A face, a mountain?

II. Solution: Piecewise parametric curves

Model arbitrary shapes with piecewise parametric curves
Each piece locally approximates part of a complex shape

III. Continuity: parametric and geometric

How to join piecewise sections? How to keep the overall curve smooth?
Continuity at joining points, or knots
Ck continuity – continuity of parametric curve and derivatives
C0 – matching position
C1 – matching position and direction (tangent), and speed
C2 – matching position, direction and curvature
Given f, g adjacent piecewise curves, we have for derivative i that
$f^{(i)}(t_k) = g^{(i)}(t_k)$ at t_k in knot k

Gk continuity – geometric continuity without respect to arc length
G0 – matching position
G1 – matching position and direction (tangent), not always speed
G2 – matching position, direction and curvature
Given f, g adjacent piecewise curves, we have for derivative i that
$f^{(i)}(t_k) = sg^{(i)}(t_k)$ at t_k in knot k, eg, the derivatives are proportional

IV. Linear, quadratic and cubic curves

Options for piecewise curves are linear, quadratic, cubic and higher order curves

- Piecewise linear approximation – commonly used
- Quadratic curve – used, but hard to get C1 continuity at both ends
- Cubic curve – has inflection point, so can switch direction and achieve C1/C2
  Used in industrial design – generally smooth enough for perception
  Not smooth enough for all applications – motion control, other

Conclusion: cubic curve is adequate for most purposes

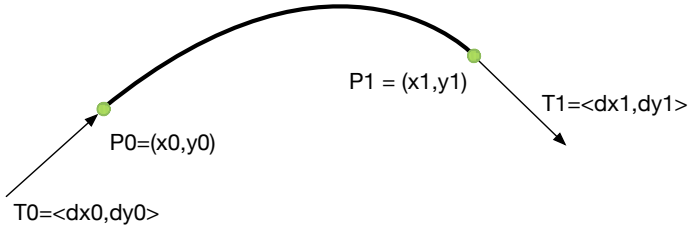
V. How to specify curves: control points and interpolation

Control point – data point that controls shape of curve
Interpolation – curve goes through control point
Approximation – curve goes close to, or is controlled by, control point
Local vs. global control – Local control, only nearby points control shape
Global control, all points control entire curve

VI. First example: Hermite cubic curve (2D version)

Given a parametric equation of (x,y) in t in [0,1]

$$x(t) = at^3 + bt^2 + ct + d$$
$$y(t) = et^3 + ft^2 + gt + h$$



The resulting curve interpolates points P0, P1 *and* matches tangents T0, T1

Since x(t) and y(t) are independent, consider first x(t) and four parameters a, b, c and d. (Developments for y(t), and z(t), are similar.) We have the derivative:

$$x'(t) = 3at^2 + 2bt + c$$

And we have the following four constraints at t = 0 (at P0) and t = 1 (at P1):

$$x(0) = d$$
$$x'(0) = c$$
$$x(1) = a + b + c + d$$
$$x'(1) = 3a + 2b + c$$

If we substitute and solve for a, b, c and d we get this curve that satisfies the constraints.

$$d = x0$$
$$c = dx0$$
$$b = -3x0 + 3x1 - 2dx0 - dx1$$
$$a = 2x0 - 2x1 + dx0 + dx1$$

Call this Hermite version 1. There will be similar equations for y(t) and z(t).

$$x(t) = (2x0 - 2x1 + dx0 + dx1)t^3 + (-3x0 + 3x1 - 2dx0 - dx1)t^2 + (dx0)t + x0$$

VII. Matrix form of curves

The development of the Hermite curve equations is fine, but can be more efficiently (and effectively) given in matrix format. If we rewrite the four constraint equations in matrix format

$$x(0) = d$$
$$x'(0) = c$$
$$x(1) = a + b + c + d$$
$$x'(1) = 3a + 2b + c$$

we get

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} x0 \\ x1 \\ dx0 \\ dx1 \end{bmatrix}$$

or

$$MA = G$$

with M the matrix, A the parameter vector, and G the geometry vector. Solving we get

$$A = M^{-1}G$$

or the Hermite basis matrix

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ dx0 \\ dx1 \end{bmatrix}$$

Call this Hermite version 2, matrix form. The basis matrix M^{-1} is a compact representation.

IIX. Vector matrix form

To make the matrix representation more compact we replace the scalar x0, x1, etc, with point and vectors P0 = <x0,y0,z0>, P1 = <x1,y1,z1>, T0=<dx0,dy0,dz0>, T1=<dx1,dy1,dz1>, to get:

$$\begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \boldsymbol{c} \\ \boldsymbol{d} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{P0} \\ \boldsymbol{P1} \\ \boldsymbol{T0} \\ \boldsymbol{T1} \end{bmatrix}$$

Now we have a polynomial with vector coefficients. Call this Hermite version 3.

$$P(t) = \boldsymbol{a}t^3 + \boldsymbol{b}t^2 + \boldsymbol{c}t + \boldsymbol{d}$$

There's one additional representation

$$P(t) = [t^3 \quad t^2 \quad t \quad 1]\begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{b} \\ \boldsymbol{c} \\ \boldsymbol{d} \end{bmatrix} = [t^3 \quad t^2 \quad t \quad 1]\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} \boldsymbol{P0} \\ \boldsymbol{P1} \\ \boldsymbol{T0} \\ \boldsymbol{T1} \end{bmatrix}$$

IX. Blending functions.

A fourth version of the Hermite equation rewrites it in terms of blending functions that weights each of the data points. We take the first version

$$x(t) = (2x0 - 2x1 + dx0 + dx1)t^3 + (-3x0 + 3x1 - 2dx0 - dx1)t^2 + (dx0)t + x0$$

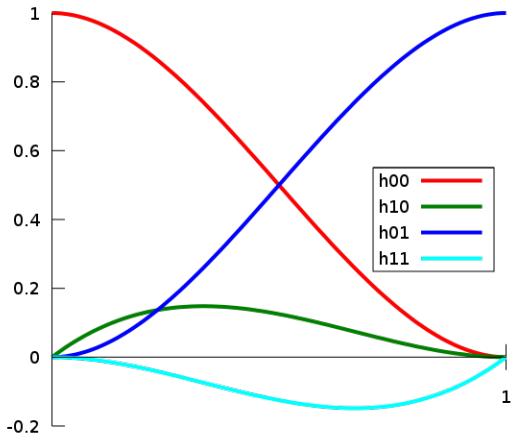
and group the data terms

$$x(t) = (2t^3 - 3t^2 + 1)x0 + (-2t^3 + 3t^2)x1 + (t^3 - 2t^2 + t)dx0 + (t^3 - t^2)dx1$$

To get the four blending functions

$$h00(t) = (2t^3 - 3t^2 + 1)$$
$$h01(t) = (-2t^3 + 3t^2)$$
$$h10(t) = (t^3 - 2t^2 + t)$$
$$h00(t) = (t^3 - t^2)$$

From Wikipedia, the four blending functions graph as below. These four functions sum to 1, so represent an affine combination.



The way to look at this is, each of the curves weight the input data points, so the red curve (h00) represents the proportion of P0 that contributes to the output point.

X. Computing the tangents

If we're given four points, we can set up a matrix to compute the tangents from four original points. Assuming these are x_{-1}, x_0, x_1, and x_2, we can define the matrix H that takes the original points into the point/tangent vector, as in $G = HP$, where P is a vector of the original points.

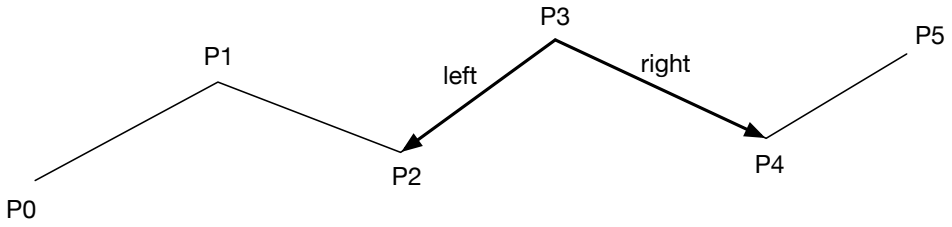
This can be inserted into the Hermite equation

Given our final Hermite equation version 5, our final version:

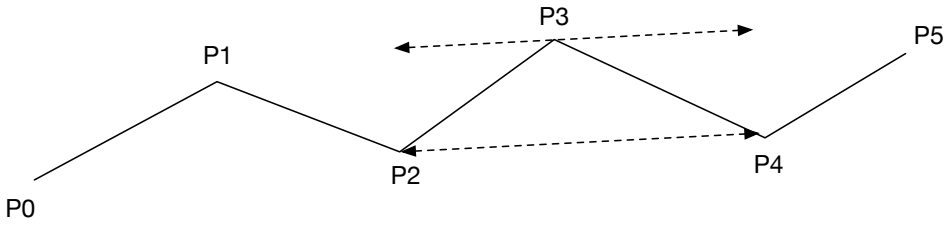
$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_{-1} \\ x_2 \end{bmatrix}$$

IIX. Catmull-Rom splines

If you build a Hermite curve from a polyline you can have a problem with C1 continuity, as the obvious way to define tangents from point to point is not symmetric. At one control point the left and right tangents would not be equal. At control point P3 in the diagram below the left and right tangents are computed from P2 and P4, respectively.



Catmull-Rom curves use a symmetric definition of the tangent at a control point by defining the slope at point Pi as the difference between P(i+1) and P(i-1).



Our previous work with Hermite curves gives us most of the required math. The key is changing the tangent matrix H which computes derivatives at control points P0 and P1. (The ½ comes from weighting the tangent vector – we used a weight of 1 for Hermite curves, but it's traditional to use ½ for Catmull-Rom. In the general development you'll see a tuning factor of α.)

$$\begin{bmatrix} x0 \\ x1 \\ dx0' \\ dx1' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1/2 & 0 & -1/2 & 0 \\ -1/2 & 0 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_{-1} \\ x_2 \end{bmatrix}$$

We use the same basis matrix as from the Hermite curves to get:

or

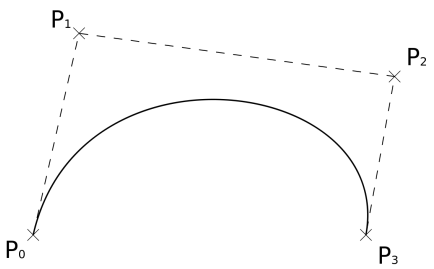
$$P(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} 2 & -2 & -0.5 & 0.5 \\ -3.5 & 3 & 1 & -0.5 \\ 0.5 & 0 & 0.5 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_{-1} \\ x_2 \end{bmatrix}$$

XII. Bezier curves

Finally, Bezier curves. These are often used graphics design, such as in programs like Adobe Illustrator. TrueType fonts use quadratic Bezier curves while Postscript and SVG use cubic. Bezier curves are deemed more flexible and intuitive, particularly for a sequence of points in a polyline, and can more easily manage kinks.

The primary difference with Hermite is that the two middle points define the tangents, and the two end points are interpolated (in the four point version.) For an extended polyline a Bezier curve may not interpolated individual points beyond the first and final.

The curve enters P0 on a tangent P0-P1, and exits P3 on a tangent P2-P3.



The translate to our Hermite curve is given by:

P0 = P0
P1 = P3
T0 = 3(P1-P0)
T1 = 3(P3-P2)

The development of the Bezier curve can be done in the same fashion as Hermite from here.

XIII. Defining Bezier curves with De Casteljau's algorithm

See PowerPoint slides on this construction.

Links

http://graphics.stanford.edu/courses/cs148-09/lectures/splines.pdf

https://pomax.github.io/bezierinfo/