Parametric Cubic Curves

- Defining them
- Connecting them
- Transforming them
- Drawing them

Hearn & Baker, pp. 315-355 (study)

Types of Curves

- Explicit

$$y = mx + b \qquad r = A_r x + B_r y + C_r$$

- Implicit

$$Ax + By + C = 0$$

$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0$$

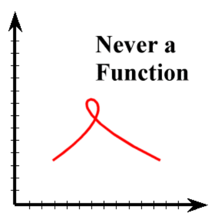
- Parametric

$$x = x_0 + (x_1 - x_0)\, t \qquad x = x_0 + r\cos\theta$$

$$y = y_0 + (y_1 - y_0)\, t \qquad y = y_0 + r\sin\theta$$

Why Parametric?

- Parametric curves are very flexible
- They are not required to be functions
  Curves can be multi-valued with respect to any coordinate system



- Parameter count generally gives the objects's dimension

$$(x(u,v), y(u,v), z(u,v))$$

- Decouples dimension of object from the dimension of the space

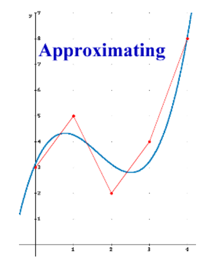
Specifying Curves

Control Points - a set of points that influence the curve's shape

Knots - control points that lie on the curve

Interpolating spline - curve passes through the control points knots

Approximating spline - control points merely influence shape

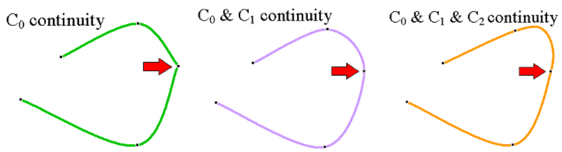


Piecewise Curve Segments

Often we will want to represent a curve as a series of curves pieced together

But we will want these curves to fit together reasonably...

Continuity-



Parametric Cubic Curves

In order to assure C_2 continuity our functions must be of at least degree 3

Here's what a parametric cubic spline function looks like:

$$x = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y = a_y t^3 + b_y t^2 + c_y t + d_y$$

Alternatively, it can be written in matrix form:

$$\begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

Solving for Coefficients

The whole story of polynomial slines is deriving their coefficients

How?

By satisfying the constraints set by the knots and continuity conditions

An Illustrative Example

Cubic Hermite Splines:

Hermite Specification

The Gradient of a Cubic Spline

$$\begin{bmatrix} \frac{dx}{dt} & \frac{dy}{dt} \end{bmatrix} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

The Hermite Specification as a Matrix Equation

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

Solve for the Hermite Coefficients

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix} = \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

Spline Basis and Geometry Matrices

$$\underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}} = \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

Resulting Cubic Hermite Spline Equation

$$\begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}}$$

Hermite Spline Demonstration

Discussion:

- Is the gradient *really* an intuitve control?
- Piecewise issues:
  - C_0 easy
  - C_1 resonable
  - C_2 tricky

Another Way to Think About Splines

The contribution of each geometric factor can be considered separately, this approach gives a so-called *blending function* associated with each factor.

BTW, this is the way I naturally think about splines

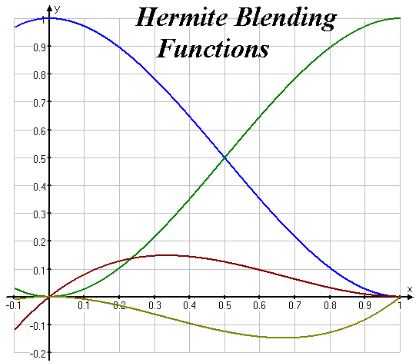
Beginning with our spline formulation:

$$\begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}}$$

By reording our multiplications we get:

$$p(t) = \begin{bmatrix} 2t^3 - 3t^2 + 1 \\ -2t^3 + 3t^2 \\ t^3 - 2t^2 + t \\ t^3 - t^2 \end{bmatrix}^{\mathbf{T}} \begin{bmatrix} p_1 \\ p_2 \\ \nabla p_1 \\ \nabla p_2 \end{bmatrix}$$

Hermite Blending Functions

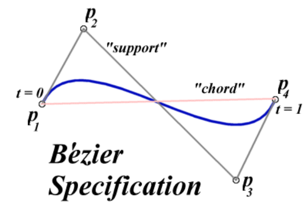


There is considerable intution hidden in these functions

One More Example

As we saw Cubic Hermite Splines presented some user friendlyness problems

Next we will define a new Spline class that has more intuitive controls



A Cubic Bezier Spine has four control points, two of which are knots.

Coefficients for Cubic Bezier Splines

It just so happens that the knot gradients of a Bezier Spline can be expressed in terms of the adjacent control points:

$$\nabla p_1 = 3(p_2 - p_1)$$

$$\nabla p_4 = 3(p_4 - p_3)$$

Note: this approach is entirely synthetic!

Using the vector is reasonable, but what makes 3 a magic number?

Actually it is, but that's a topic for another time!

Here's the Trick!

Knowing this we can formulate a Bezier spline in terms of the Hermite geometry spec

$$\underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \frac{dx_1}{dt} & \frac{dy_1}{dt} \\ \frac{dx_2}{dt} & \frac{dy_2}{dt} \end{bmatrix}}_{\mathbf{G}_{Hermite}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

And substituting gives:

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix}}_{\mathbf{M}_{Hermite}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

Basis and Geometry Matrices for Bezier Splines

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{Bezier}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{Bezier}}$$

While cute, this derivation was not very natural

Bezier Blending Functions

The reasonable justification for Bezier spline basis can only be approached by considering its blending functions:

$$p(t) = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}^{\mathbf{T}} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

This family of polynomials (called order-3 Bernstein polynomials) have the following unique properties:

- They are all positive in the interval [0, 1]
- Thier sum is equal to 1

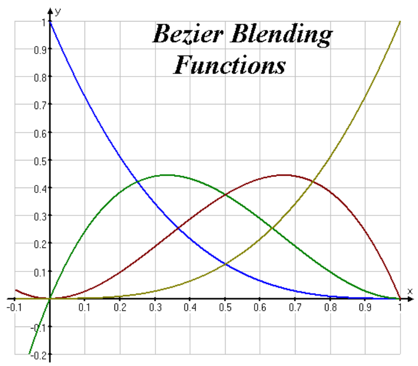
Plots of Bezier Blending Functions

Thus, every point on the curve is a linear combination of the control points.

The weights of this combination are all positive.

The sum of these weights is 1.

Therefore, the curve is a *convex combination* of the control points!



Bezier Demonstration

Discussion:

- Strange mix of points on and off the curve
- Piecewise issues:
  - C_0 easy
  - C_1 easy
  - C_2 easy

Usage notes:

Clicking the right mouse will recursively sub-divide the support structure.

Other Cubic Spline Types:

We can specify the slope at knots points using the control points on either side, and parameterize the wieghting factors:

$$\nabla p_2 = s(p_3 - p_1)$$
$$\nabla p_3 = s(p_4 - p_2)$$

Repeating the same derivation gives:

$$\begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \underbrace{\begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}}_{\mathbf{M}_{cardinal}} \underbrace{\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}}_{\mathbf{G}_{cardinal}}$$

Cardinal Spline Demo

Discussion:

- The version with s = 1/2 is called a Catmull-Rom interpolationing spline
- Piecewise issues:
  - C_0 automatic
  - C_1 automatic
  - C_2 automatic
  - *Strange* control
  - Needs 2 extra control points

Usage notes:

Clicking the right mouse will recursively sub-divide the support structure.

Uniform B-Splines

Discussion:

- This (and important variants on it) are by far the most popular spline basis.
- We'll return to these later when we consider 3-D modeling primitives
- Piecewise issues:
  - C_0 automatic
  - C_1 automatic
  - C_2 automatic
  - Very flexible control
  - Needs 2 extra control points

How to write all these Spline Rendering Routines?

Consider the spline basis formulations for two spline types:

$$P = T \times M_{spline_1} \times G_{spline_1}$$

$$P = T \times M_{spline_2} \times G_{spline_2}$$

$$T \times M_{spline_1} \times G_{spline_1} = T \times M_{spline_2} \times G_{spline_2}$$

We can transform the control points from one spline basis to another:

$$M_{spline_1} \times G_{spline_1} = M_{spline_2} \times G_{spline_2}$$

$$G_{spline_1} = M_{spline_1}^{-1} \times M_{spline_2} \times G_{spline_2}$$

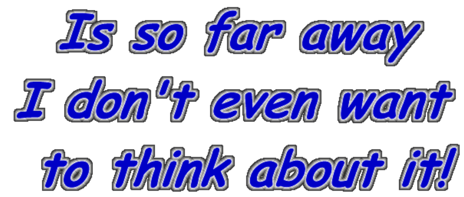
Rasterizing Code

All we need is code to render one type of spline!

I choose to use the Bezier spline because of it's convexity property allows me to write a simple recursive algorithm:

```
public void spline(ControlPoint p0,
                   ControlPoint p1,
                   ControlPoint p2,
                   ControlPoint p3, int pix) {
    float len = ControlPoint.dist(p0,p1)
              + ControlPoint.dist(p1,p2)
              + ControlPoint.dist(p2,p3);
    float chord = ControlPoint.dist(p0,p3);
    if (Math.abs(len - chord) < 0.25f) return;
    fatPixel(pix, p0.x, p0.y);
    ControlPoint p11 = ControlPoint.midpoint(p0, p1);
    ControlPoint tmp = ControlPoint.midpoint(p1, p2);
    ControlPoint p12 = ControlPoint.midpoint(p11, tmp);
    ControlPoint p22 = ControlPoint.midpoint(p2, p3);
    ControlPoint p21 = ControlPoint.midpoint(p22, tmp);
    ControlPoint p20 = ControlPoint.midpoint(p12, p21);
    spline(p20, p12, p11, p0, pix);
    spline(p3, p22, p21, p20, pix);
}
```


Next Time



(Project #2 is due soon... Have a happy Columbus day)