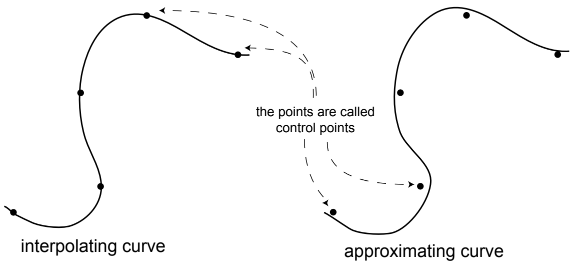
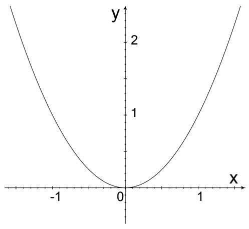
Chapter 14

Spline Curves

A *spline curve* is a mathematical representation for which it is easy to build an interface that will allow a user to design and control the shape of complex curves and surfaces. The general approach is that the user enters a sequence of points, and a curve is constructed whose shape closely follows this sequence. The points are called *control points.* A curve that actually passes through each control point is called an *interpolating curve*; a curve that passes near to the control points but not necessarily through them is called an *approximating curve.*



Once we establish this interface, then to change the shape of the curve we just move the control points.



The easiest example to help us to understand how this works is to examine a curve that is like the graph of a function, like $y = x^2$. This is a special case of a polynomial function.

14.1 Polynomial curves

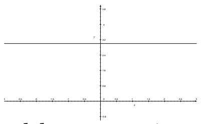
Polynomials have the general form:

$$y = a + bx + cx^2 + dx^3 + \dots$$

The *degree* of a polynomial corresponds with the highest coefficient that is non-zero. For example if c is non-zero but coefficients d and higher are all zero, the polynomial is of degree 2. The shapes that polynomials can make are as follows:

degree 0: *Constant*, only a is non-zero.

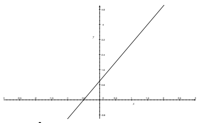
Example: $y = 3$



A constant, uniquely defined by one point.

degree 1: *Linear*, b is highest non-zero coefficient.

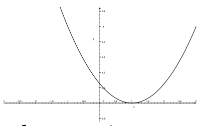
Example: $y = 1 + 2x$



A line, uniquely defined by two points.

degree 2: *Quadratic*, c is highest non-zero coefficient.

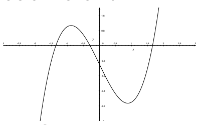
Example: $y = 1 - 2x + x^2$



A parabola, uniquely defined by three points.

degree 3: *Cubic*, d is highest non-zero coefficient.

Example: $y = -1 - 7/2x + 3/2x^3$

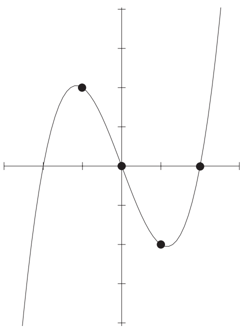


A cubic curve (which can have an inflection, at $x = 0$ in this example), uniquely defined by four points.

The degree three polynomial – known as a *cubic polynomial* – is the one that is most typically chosen for constructing smooth curves in computer graphics. It is used because

1. it is the lowest degree polynomial that can support an inflection – so we can make interesting curves, and
2. it is very well behaved numerically – that means that the curves will usually be smooth like this: and not jumpy like this: .

So, now we can write a program that constructs cubic curves. The user enters four control points, and the program solves for the four coefficients a, b, c and d which cause the polynomial to pass through the four control points. Below, we work through a specific example.



Typically the interface would allow the user to enter control points by clicking them in with the mouse. For example, say the user has entered control points $(-1, 2), (0, 0), (1, -2), (2, 0)$ as indicated by the dots in the figure to the left.

Then, the computer solves for the coefficients a, b, c, d and might draw the curve shown going through the control points, using a loop something like this:

```
glBegin(GL_LINE_STRIP);
   for(x = -3; x <= 3; x += 0.25)
      glVertex2f(x, a + b * x + c * x * x + d * x * x * x);
glEnd();
```

Note that the computer is not really drawing the curve. Actually, all it is doing is drawing straight line segments through a sampling of points that lie on the curve. If the sampling is fine enough, the curve will appear to the user as a continuous smooth curve.

The solution for a, b, c, d is obtained by simultaneously solving the 4 linear equations below, that are obtained by the constraint that the curve must pass through the 4 points:

general form: $a + bx + cx^2 + dx^3 = y$

point $(-1, 2)$**:** $a - b + c - d = 2$

point $(0, 0)$**:** $a = 0$

point $(1, -2)$**:** $a + b + c + d = -2$

point $(2, 0)$**:** $a + 2b + 4c = 8d = 0$

This can be written in matrix form

$$M\mathbf{a} = \mathbf{y},$$

or (one row for each equation)

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -2 \\ 0 \end{bmatrix}$$

The solution is

$$\mathbf{a} = M^{-1}\mathbf{y},$$

which is hard to do on paper but easy to do on the computer using matrix and vector routines.

For example, using the code I have provided, you could write:

```
Vector a(4), y(4);
Matrix M(4, 4);
y[0] = 2; y[1] = 0; y[2] = -2; y[3] = 0;
// fill in all rows of M
M[0][0] = 1; M[0][1] = -1; M[0][2] = 1; M[0][3] = -1;
// etc.  to fill all 4 rows
a = M.inv() * y;
```

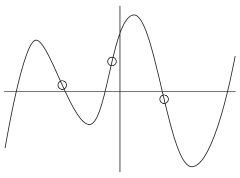
After this computation, `a[0]` contains the value of a, `a[1]` of b, `a[2]` of c and `a[3]` of d. For this example the correct values are $a = 0$, $b = -2\frac{2}{3}$, $c = 0$, and $d = \frac{2}{3}$.

14.2 Piecewise polynomial curves

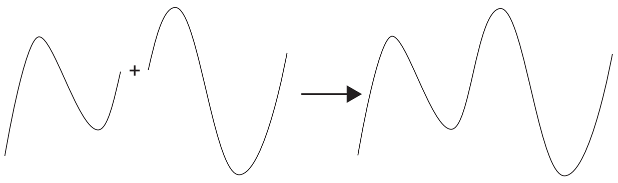
In the previous section, we saw how four control points can define a cubic polynomial curve, allowing the solution of four linear equations for the four coefficients of the curve. Here we will see how more complex curves can be made using two new ideas:

1. construction of piecewise polynomial curves,
2. parameterization of the curve.

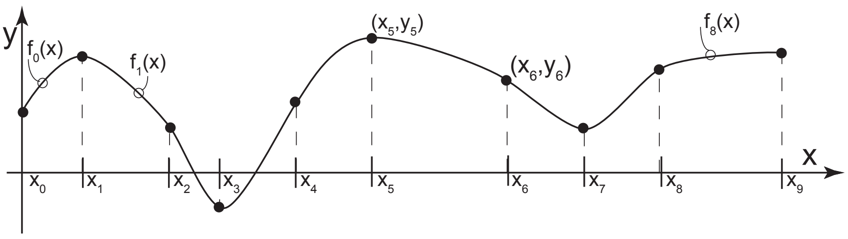
Suppose we wanted to make the curve shown to the right. We know that a single cubic curve can only have one inflection point, but this curve has three, marked with O's. We could make this curve by entering extra control points and using a 5th degree polynomial, with six coefficients, but polynomials with degree higher than three tend to be very sensitive to the positions of the control points and thus do not always make smooth shapes.



The usual solution to this problem in computer graphics and computer aided design is to construct a complex curve, with a high number of inflection points, by piecing together several cubic curves:



Here is one way that this can be done. Let each pair of control points represent one segment of the curve. Each curve segment is a cubic polynomial with its own coefficients:

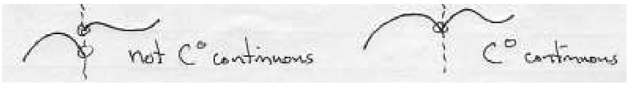


In this example, the ten control points have ascending values for the x coordinate, and are numbered with indices 0 through 9. Between each control point pair is a function, which is numbered identically to the index of its leftmost point. In general, $f_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$ is the function representing the curve between control points i and $i + 1$.

Because each curve segment is represented by a cubic polynomial function, we have to solve for four coefficients for each segment. In this example we have $4 \times 9 = 36$ coefficients to solve for. How shall we do this?

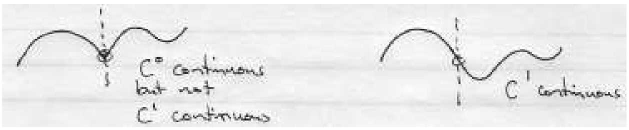
Here is one way:

1. We require that each curve segment pass through its control points. Thus, $f_i(x_i) = y_i$, and $f_i(x_{i+1}) = y_{i+1}$. *This enforces C^0 continuity – that is, where the curves join they meet each other.*

   Note that for each curve segment this gives us two linear equations: $a_i + b_ix_i + c_ix_i^2 + d_ix_i^3 = y_i$, and $a_i + b_ix_{i+1} + c_ix_{i+1}^2 + d_ix_{i+1}^3 = y_{i+1}$. But to solve for all four coefficients we need two more equations for each segment.

2. We require that the curve segments have the same slope where they join together. Thus, $f_i'(x_{i+1}) = f_{i+1}'(x_{i+1})$. *This enforces C^1 continuity – that is that slopes match where the curves join.*

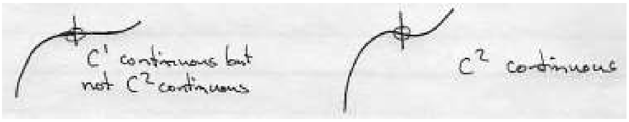
   

   Note that: $f_i(x) = a_i + b_ix + c_ix^2 + d_ix^3$, so that $f_i'(x) = b_i + 2c_ix + 3d_ix^2$.

   And C^1 continuity gives us one more linear equation for each segment $b_i + 2c_ix_{i+1} + 3d_ix_{i+1}^2 = b_{i+1} + 2c_{i+1}x_{i+1} + 3d_{i+1}x_{i+1}^2$ or $b_i + 2c_ix_{i+1} + 3d_ix_{i+1}^2 - b_{i+1} - 2c_{i+1}x_{i+1} - 3d_{i+1}x_{i+1}^2 = 0$.

3. To get the fourth equation we require that the curve segments have the same curvature where they join together. Thus, $f_i''(x_{i+1}) = f_{i+1}''(x_{i+1})$.

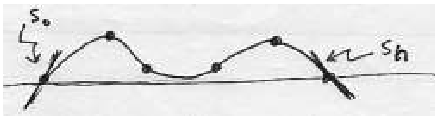
   This enforces C^2 continuity – that curvatures match at the join.

   Now: $f''i(x) = 2c_i + 6d_ix$

   And C^2 continuity gives us the additional linear equation that we need $2c_i + 6d_ix_{i+1} = 2c_{i+1} + 6d_{i+1}x_{i+1}$ or $2c_i + 6d_ix_{i+1} - 2c_{i+1} - 6d_{i+1}x_{i+1} = 0$

4. Now we are almost done, but you should note that at the left end of the curve we are missing the C^1 and C^2 equations since there is no segment on the left. So, we are missing two equations needed to solve the entire system. We can get these by having the user supply the slopes at the two ends. Let us call these slopes s_0 and s_n.

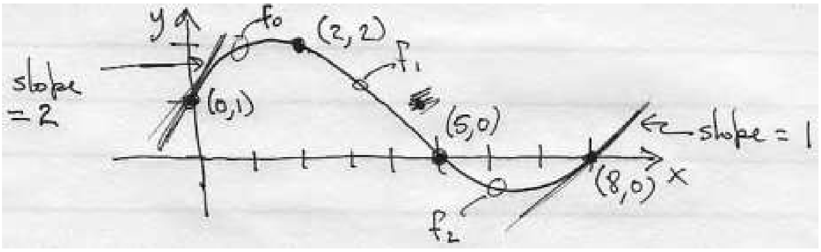
   This gives $f_0'(x_0) = s_0$, and $f_{n-1}'(x_n) = s_n$ so that $b_0 + c_0x_0 + 2d_0x_0^2 = s_0$, and $b_{n-1} + c_{n-1}x_n + 2d_{n-1}x_n^2 = s_n$.

As we did with the case of a single cubic spline, we have a set of linear equations to solve for a set of unknown coefficients. Once we have the coefficients we can draw the curve a segment at a time. Again the equation to solve is

$$M\mathbf{a} = \mathbf{y} \implies \mathbf{a} = M^{-1}\mathbf{y},$$

where each row of the matrix M is taken from the left-hand side of the linear equations, the vector $\mathbf{a}$ is the set of coefficients to solve for, and the vector $\mathbf{y}$ is the set of known right-hand values for each equation.

Following is a worked example with three segments:



Segment 0: $f_0(x)$	
Slope at 0:	$b_0 = 2$
Curve through (0, 1):	$a_0 = 1$
Curve through (2, 2):	$a_0 + 2b_0 + 4c_0 + 8d_0 = 2$
Slopes match at join with f_1:	$b_0 + 4c_0 + 12d_0 - b_1 - 4c_1 - 12d_1 = 0$
Curvatures match at join with f_1:	$2c_0 + 12d_0 - 2c_1 - 12d_1 = 0$

Segment 1: $f_1(x)$	
Curve through (2, 2):	$a_1 + 2b_1 + 4c_1 + 8d_1 = 2$
Curve through (5, 0):	$a_1 + 5b_1 + 25c_1 + 125d_1 = 0$
Slopes match at join with f_2:	$b_1 + 10c_1 + 75d_1 - b_2 - 10c_2 - 75d_2 = 0$
Curvatures match at join with f_2:	$2c_1 + 30d_1 - 2c_2 - 30d_2 = 0$

Segment 2: $f_2(x)$	
Curve through (5, 0):	$a_2 + 5b_2 + 25c_2 + 125d_2 = 0$
Curve through (8, 0):	$a_2 + 8b_2 + 64c_2 + 512d_2 = 0$
Slope at 8:	$b_2 + 16c_2 + 192d_2 = 1$

$$M\mathbf{a} = \mathbf{y}, \text{ or}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 4 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 12 & 0 & -1 & -4 & -12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 12 & 0 & 0 & -2 & -12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 5 & 25 & 125 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 10 & 75 & 0 & -1 & -10 & -75 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 30 & 0 & 0 & -2 & -30 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 & 25 & 125 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 8 & 64 & 512 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 8 & 128 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ d_0 \\ a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 2 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

so

$$\mathbf{a} = M^{-1}\mathbf{y},$$

Again, this is hard to solve by hand but easy on the computer.

14.3 Curve parameterization

So far we have learned how a sequence of control points can define a piecewise polynomial curve, using cubic functions to define curve segments between control points and enforcing various levels of continuity where segments join. In particular, we employed

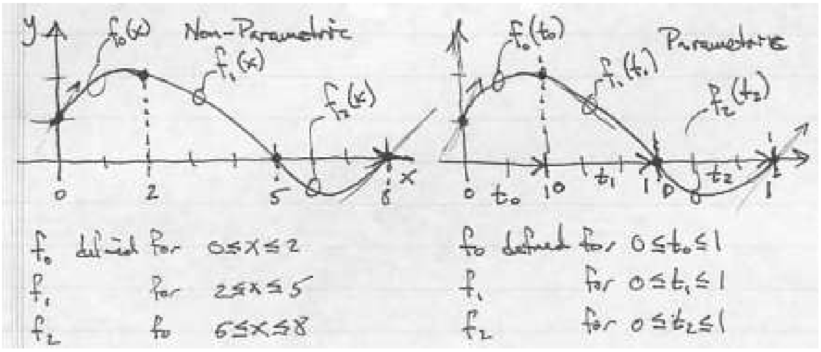
- C^0 continuity, meaning that the two segments match values at the join.
- C^1 continuity, meaning that they match slopes at the join.
- C^2 continuity, meaning that they match curvatures at the join.

We were able to determine coefficients for the curve segments via a set of linear equations

$$M\mathbf{a} = \mathbf{y} \implies \mathbf{a} = M^{-1}\mathbf{y},$$

where $\mathbf{a}$ is the vector of all coefficients, $\mathbf{y}$ is the vector of constants on the right-hand side of the linear equations, and M is a matrix encoding the C^0, C^1 and C^2 conditions.

This approach can be modified to specify each curve segment in parametric form, as indicated in the figure below.


Non-Parametric
$f_0(x)$
$f_1(x)$
$f_2(x)$
0
2
5
8
x
f_0 defined for $0 \le x \le 2$
f_1 for $2 \le x \le 5$
f_2 for $5 \le x \le 8$
Parametric
$f_0(t_0)$
$f_1(t_1)$
$f_2(t_2)$
0
t_0
1 0
t_1
1 0
t_2
1
f_0 defined for $0 \le t_0 \le 1$
f_1 for $0 \le t_1 \le 1$
f_2 for $0 \le t_2 \le 1$

In the example, both curves are identical, however, the equations describing them will be different. In the parametric form on the right, we have defined parameters t_0, t_1 and t_2 that vary between 0 and 1 as we step along the x axis between control points. We could write equations:

$$\begin{aligned} t_0 &= x/2, &&\text{with } \frac{dt_0}{dx} = 1/2 \\ t_1 &= (x-2)/3, &&\text{with } \frac{dt_1}{dx} = 1/3 \\ t_2 &= (x-5)/3, &&\text{with } \frac{dt_2}{dx} = 1/3 \end{aligned}$$

relating the t's to the original x coordinate. The derivatives indicate how quickly each t varies as we move in the x direction.

Now we specify each curve segment by a parametric cubic curve

$$f_i(t_i) = a_i + b_i t_i + c_i t_i^2 + d_i t_i^3$$

At the left side of segment i, $t_i = 0$ and at the right $t_i = 1$, so C^0 continuity is simply

$$\begin{aligned} f_i(0) &= a_i = y_i \\ f_i(1) &= a_i + b_i + c_i + d_i = y_{i+1} \end{aligned}$$

Notice, that in this form the a_i coefficients are simply the y coordinates of the ith control points, and do not have to be solved for.

For C^1 continuity we differentiate once with respect to x using the chain rule:

$$D_x f_i = \frac{\partial f_i}{\partial t_i}\frac{dt_i}{dx} = f_i^{'} \frac{1}{x_{i+1} - x_i},$$

and for C^2 continuity we differentiate twice:

$$D_x^2 f_i = \left(\frac{\partial f_i^{'}}{\partial t_i}\frac{dt_i}{dx}\right)\frac{dt_i}{dx} = f_i^{''} \frac{1}{(x_{i+1} - x_i)^2}.$$

It is important that these quantities be calculated in spatial (i.e. x), not in parametric coordinates, since we want the curves to join smoothly in space, not with respect to our arbitrary parameterization.

We force f_i and f_{i+1} to match slopes by:

$$(D_x f_i)(1) = (D_x f_{i+1})(0),$$

or

$$(b_i + 2c_i + 3d_i)\frac{1}{x_{i+1} - x_i} = b_{i+1}\frac{1}{x_{i+2} - x_{i+1}}, \quad \text{or finally}$$

$$b_i + 2c_i + 3d_i - \frac{x_{i+1} - x_i}{x_{i+2} - x_{i+1}} b_{i+1} = 0$$

We force f_i and f_{i+1} to match curvatures by:

$$(D_x^2 f_i)(1) = (D^2 f_{i+1})(0)$$

or

$$(2c_i + 6d_i)\frac{1}{(x_{i+1} - x_i)^2} = 2c_{i+1}\frac{1}{(x_{i+2} - x_{i+1})^2}, \quad \text{or finally}$$

$$2c_i + 6d_i - 2\frac{(x_{i+1} - x_i)^2}{(x_{i+2} - x_{i+1})^2} c_{i+1} = 0.$$

Remember that we provided two extra equations by specifying slopes at the 2 endpoints s_0 and s_n. So this gives

$$\frac{f_0'(0)}{x_1 - x_0} = s_0, \quad \frac{f_{n-1}'(1)}{x_n - x_{n-1}} = s_n,$$

or

$$b_0 = (x_1 - x_0)s_0,$$

and

$$b_{n-1} + 2c_{n-1} + 3d_{n-1} = (x_n - x_{n-1})s_n.$$

Looking back at our previous example, which we did without parameterization, we can rewrite all of our equations, and the final linear system that we have to solve. On big advantage of this approach is that we reduce one row and one column from the matrix for each control point, since we already know the values of all of the a coefficients.

Segment 0: $f_0(t_0)$	
Slope at 0:	$b_0 = (2-0)2 = 4$
Curve through (0, 1):	$a_0 = 1$
Curve through (2, 2):	$a_0 + b_0 + c_0 + d_0 = 2 or b_0 + c_0 + d_0 = 2 - 1 = 1$
Slopes match:	$b_0 + 2c_0 + 3d_0 - (\frac{2-0}{5-2})b_1 = 0$
Curvatures match:	$2c_0 + 6d_0 - 2(\frac{2}{3})^2 c_1 = 0$

Segment 1: $f_1(t_1)$	
Curve through (2, 2):	$a_1 = 2$
Curve through (5, 0):	$b_1 + c_1 + d_1 = 0 - 2 = -2$
Slopes match:	$b_1 + 2c_1 + 3d_1 - (\frac{8-5}{5-2})b_2 = 0$
Curvatures match:	$2c_1 + 6d_1 - 2c_2 = 0$

Segment 2: $f_2(t_2)$	
Curve through (5, 0):	$a_2 = 0$
Curve through (8, 0):	$b_2 + c_2 + d_2 = 0 - 0 = 0$
Slope at 8:	$b_2 + 2c_2 + 3d_2 = (8-5)2 = 6$

$$M\mathbf{a} = \mathbf{y}, \quad \text{or}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & -\frac{2}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & -\frac{8}{9} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 3 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 6 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} b_0 \\ c_0 \\ d_0 \\ b_1 \\ c_1 \\ d_1 \\ b_2 \\ c_2 \\ d_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 0 \\ 0 \\ -2 \\ 0 \\ 0 \\ 0 \\ 6 \end{bmatrix},$$

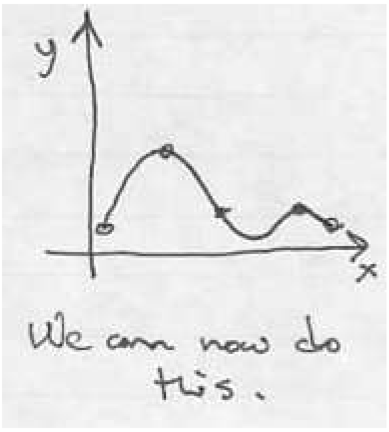
so

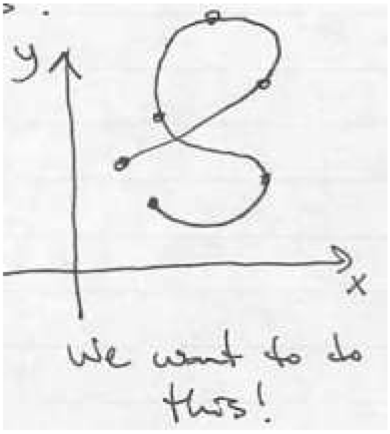
$$\mathbf{a} = M^{-1}\mathbf{y}$$

Once you have understood the notion of piecewise parameterization, the rest follows in a straightforward way.

14.4 Space curves

We now know how to make arbitrary functions from a set of control points and piecewise cubic curves, and we know how to use parameters to simplify their construction. Now we will extend these ideas to arbitrary curves in two and three dimensional space.





For the earlier examples, we used t for a parameter, but to distinguish the fact that we will now be looking at space curves, we will follow the usual notation in the literature and use the letter u.

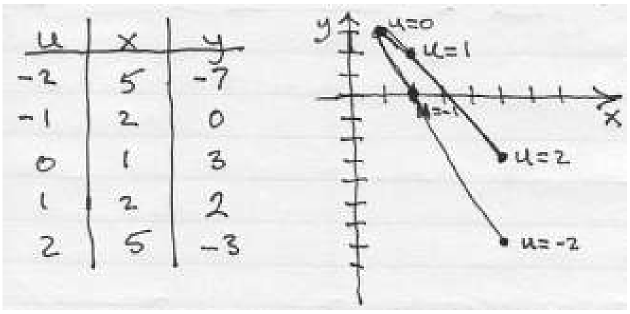
Suppose we have two functions

$$x = u^2 + 1,$$

and

$$y = -2u^2 + u + 3.$$

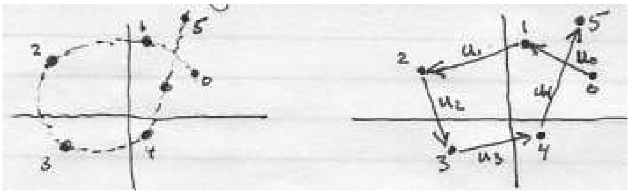
As u varies, both x and y vary. We can make a table and we can plot the points.



u	x	y
-2	5	-7
-1	2	0
0	1	3
1	2	2
2	5	-3

We see that the curve is described in 2-dimensional space, and unlike the graph of a function it can wrap back on itself. By picking appropriate functions $x = f(u)$ and $y = g(u)$ we can describe whatever 2D shape we desire.

Suppose, for example, that we have a sequence of control points. Then between each pair of points we can define two cubic curves as a function of a parameter that varies from 0 to 1 between the control points and get a smooth curve between the points by applying C^0, C^1, C^2 constraints, as indicated by the example in the following figure.



In the figure, u_1 is 0 at point 1 and 1 at point 2. A cubic curve is defined between points 1 and 2 by

$$x = a_{x1} + b_{x1}u_1 + c_{x1}u_1^2 + d_{x1}u_1^3,$$

and

$$y = a_{y1} + b_{y1}u_1 + c_{y1}u_1^2 + d_{y1}u_1^3.$$

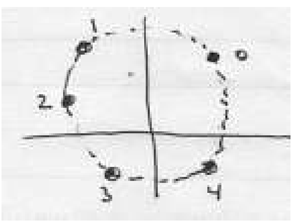
We write two systems of linear equations for the x and y coordinates separately: $M_x\mathbf{a}_x = \mathbf{x}$ and $M_y\mathbf{a}_y = \mathbf{y}$. Each system is solved following the same process we used in the previous sections; the only major difference being that we solve two linear systems instead of one.

Note that there are some differences in the details that have to be attended to. First, instead of computing the single derivative $\frac{dt_i}{dx}$ for each segment as we did above, will now need the two derivatives $\frac{\partial u_i}{\partial x}$ and $\frac{\partial u_i}{\partial y}$ for the chain rule. If we let each u_i vary linearly from 0 to 1 for a segment between its control points $\mathbf{p}_i = (x_i, y_i)$ and $\mathbf{p}_{i+1} = (x_{i+1}, y_{i+1})$ then

$$\frac{\partial u_i}{\partial x} = \frac{1}{x_{i+1} - x_i},$$

and

$$\frac{\partial u_i}{\partial y} = \frac{1}{y_{i+1} - y_i}.$$



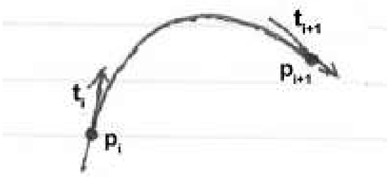
There is one more point to consider. Now that we have full space curves, it is quite possible for a curve to join back on itself to make a closed figure. If we have this situation, and we enforce C^0, C^1 and C^2 continuity where the curves join, then we have all of the equations needed without requiring the user to enter any slopes.

14.5 Interpolation methods

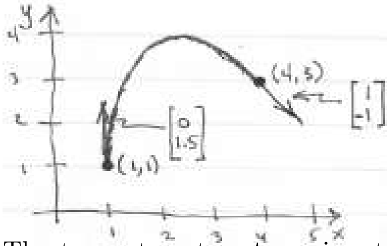
In the notes we have worked through a method for building an interpolating spline curve through a set of control points by using continuity C^0, slope C^1 and curvature C^2 constraints, where spline segments join. This is the method for computing *natural cubic splines.* It was very good for helping us to understand the basic approach to constructing piecewise cubic splines, but it is not a very good method for use in designing shapes. This is because the entire curve is determined by one set of linear equations $M\mathbf{a} = \mathbf{y}$. Therefore, moving just one control point will affect the shape of the entire curve. You can understand that if you are designing a shape, you would like to be able to have *local control* over the portion of the shape you are working on, without worry that changes you make in one place may affect distant parts of the shape.

For piecewise cubic interpolating curves, there are various ways of obtaining local shape control. The most important of these are *Hermite Splines*, *Catmull-Rom Splines*, and *Cardinal Splines.* These are explained quite well in a number of computer graphics textbooks, but let us do a few examples to illustrate these methods. Note, for each example we will be looking at only one segment of a piecewise curve. For each of these methods, each segment is calculated separately, so to get the entire curve the calculations must be repeated over all control points.

14.5.1 Hermite cubic splines



$\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ are two successive control points. For Hermite Splines the user must provide a slope at each control point, in the form of a vector $\mathbf{t}_i$, tangent to the curve at control point i. Letting parameter u_i vary from 0 at $\mathbf{p}_i$ to 1 at $\mathbf{p}_{i+1}$, we use continuity C^0 and slope C^1 constraints at the two control points.


y
4
3
2
1
(4, 3)
$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$
$\begin{bmatrix} 0 \\ 1.5 \end{bmatrix}$
(1, 1)
1
2
3
4
5
x

In the example to the left, $\mathbf{p}_i = (1,1)$, and $\mathbf{p}_{i+1} = (4,3)$, thus $\frac{du_i}{dx} = \frac{1-0}{4-1} = \frac{1}{3}$, and $\frac{du_i}{dy} = \frac{1-0}{3-1} = \frac{1}{2}$. The tangent vectors are $\mathbf{t}_i = \begin{bmatrix} 0 \\ 1.5 \end{bmatrix}$, and $\mathbf{t}_{i+1} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

The tangent vectors $\mathbf{t}$ are input by the user as direction vectors of the form

$\begin{bmatrix} \triangle x \\ \triangle y \end{bmatrix}$, but we need slopes in parametric form, so we compute the scaled tangents

$$Dp_i = \begin{bmatrix} \triangle x \frac{du_i}{dx} \\ \triangle y \frac{du_i}{dy} \end{bmatrix} = \begin{bmatrix} 0 \\ (\frac{3}{2})(\frac{1}{2}) \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{3}{4} \end{bmatrix},$$

and

$$Dp_{i+1} = \begin{bmatrix} \triangle x \frac{du_{i+1}}{dx} \\ \triangle y \frac{du_{i+1}}{dy} \end{bmatrix} = \begin{bmatrix} (1)(\frac{1}{3}) \\ (-1)(\frac{1}{2}) \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ -\frac{1}{2} \end{bmatrix}.$$

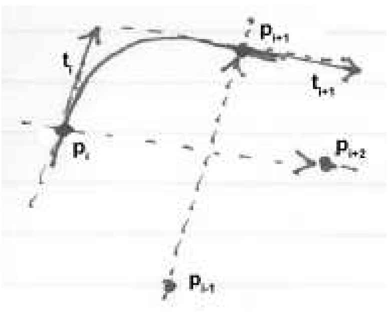
Linear system for example Hermite Spline	
Curve passes through $\mathbf{p}_i$ at $u_i = 0$:	$x(0) = a_x = 1$
	$y(0) = a_y = 1$
Curve passes through $\mathbf{p}_{i+1}$ at $u_i = 1$:	$x(1) = a_x + b_x + c_x + d_x = 4$
	$y(1) = a_y + b_y + c_y + d_y = 3$
Slope at $\mathbf{p}_i$:	$x'(0) = b_x = 0$
	$y'(0) = b_y = \frac{3}{4}$
Slope at $\mathbf{p}_{i+1}$:	$x'(1) = b_x + 2c_x + 3d_x = \frac{1}{3}$
	$y'(1) = b_y + 2c_y + 3d_y = -\frac{1}{2}$

Thus in matrix form:

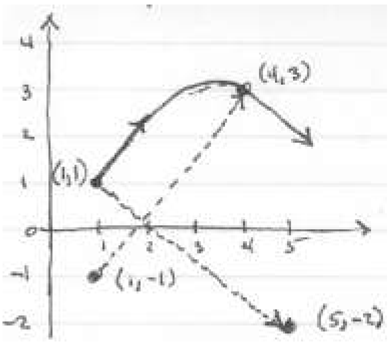
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 4 & 3 \\ 0 & \frac{3}{4} \\ \frac{1}{3} & -\frac{1}{2} \end{bmatrix}$$

Note, that the two matrices on the left-hand side of this equation are constant, and do not depend upon the control points or slopes. Only the matrix on the right varies with the configuration.

14.5.2 Catmul-Rom splines



In the case of Catmul-Rom splines, $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ are again two successive control points. Now, instead of requiring the user to enter slopes at each control point, we use the previous point $\mathbf{p}_{i-1}$ and the subsequent point $\mathbf{p}_{i+2}$ to determine slopes at $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$. The vector between $\mathbf{p}_{i-1}$ and $\mathbf{p}_{i+1}$ is used to give the slope at $\mathbf{p}_i$ and the vector between $\mathbf{p}_i$ and $\mathbf{p}_{i+2}$ to give the slope at $\mathbf{p}_{i+1}$. We set $\mathbf{t}_i = \frac{1}{2}(\mathbf{p}_{i+1} - \mathbf{p}_{i-1})$ and $\mathbf{t}_{i+1} = \frac{1}{2}(\mathbf{p}_{i+2} - \mathbf{p}_i)$.



In the example to the left, as in the previous example, $\mathbf{p}_i = (1,1)$, and $\mathbf{p}_{i+1} = (4,3)$, thus again $\frac{du_i}{dx} = \frac{1}{3}$, and $\frac{du_i}{dy} = \frac{1}{2}$. In addition, we have $\mathbf{p}_{i-1} = (1,-1)$ and $\mathbf{p}_{i+2} = (5,-2)$. Thus, the tangent vectors are $\mathbf{t}_i = \frac{1}{2}(\mathbf{p}_{i+1} - \mathbf{p}_{i-1}) = \begin{bmatrix} \frac{3}{2} \\ 2 \end{bmatrix}$, and $\mathbf{t}_{i+1} = \frac{1}{2}(\mathbf{p}_{i+2} - \mathbf{p}_i) = \begin{bmatrix} 2 \\ -\frac{3}{2} \end{bmatrix}$.

Again, the tangent vectors $\mathbf{t}$ are in the form $\begin{bmatrix} \triangle x \\ \triangle y \end{bmatrix}$, but we need slopes in parametric form, so we compute the scaled tangents

$$Dp_i = \begin{bmatrix} \triangle x \frac{du_i}{dx} \\ \triangle y \frac{du_i}{dy} \end{bmatrix} = \begin{bmatrix} (\frac{3}{2})(\frac{1}{3}) \\ (2)(\frac{1}{2}) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix},$$

and

$$Dp_{i+1} = \begin{bmatrix} \triangle x \frac{du_{i+1}}{dx} \\ \triangle y \frac{du_{i+1}}{dy} \end{bmatrix} = \begin{bmatrix} (2)(\frac{1}{3}) \\ (-\frac{3}{2})(\frac{1}{2}) \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ -\frac{3}{4} \end{bmatrix}.$$

Linear system for example Catmul-Rom Spline

Curve passes through $\mathbf{p}_i$ at $u_i = 0$:	$x(0) = a_x = 1$
	$y(0) = a_y = 1$
Curve passes through $\mathbf{p}_{i+1}$ at $u_i = 1$:	$x(1) = a_x + b_x + c_x + d$
	$y(1) = a_y + b_y + c_y + d$
Slope at $\mathbf{p}_i$:	$x'(0) = b_x = \frac{1}{2}$
	$y'(0) = b_y = 1$
Slope at $\mathbf{p}_{i+1}$:	$x'(1) = b_x + 2c_x + 3d_x$
	$y'(1) = b_y + 2c_y + 3d_y$

Thus in matrix form:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 4 & 3 \\ \frac{1}{2} & 1 \\ \frac{2}{3} & -\frac{3}{4} \end{bmatrix}$$

14.5.3 Cardinal splines

Cardinal splines generalize the Catmul-Rom splines by providing a shape parameter t. With $t = 0$ we have Catmul-Rom. Values of $t > 0$ tighten the curve, making curvatures higher at the joins and values of $t < 0$ loosen the curve. For

Cardinal Splines:

$$\mathbf{t}_i = \frac{1}{2}(1-t)(\mathbf{p}_{i+1} - \mathbf{p}_{i-1}),$$

and

$$\mathbf{t}_{i+1} = \frac{1}{2}(1-t)(\mathbf{p}_{i+2} - \mathbf{p}_{i}).$$