Curves III: Cubic Splines

In this document, we look at a number of methods for generating plane or space curves by piecing together cubic or third-degree polynomials:

$$p(u) = au^3 + bu^2 + cu + d \qquad \text{(CURV3-1)}$$

where p(u) stands for the x, y, or z coordinate of a point on the curve; the symbols a, b, c, and d stand for constants, and u is a parameter. As u runs through its defined range, the point p traverses a segment of the overall curve. Note that we will generally need a different set of values of (a, b, c, d) for each of the coordinates x, y, and z.

For the most part, the methods discussed in this document give interpolating curves – the curves pass through a given set of control points.

Because we will be joining smaller cubic curve segments together to form a longer composite curve, we cannot avoid going deeper into issues of continuity and smoothness than has been necessary in previous documents. Much of what follows will make no sense unless we review a few very basic facts from differential calculus. Even if you've never studied calculus before, take a few minutes to read the following two sections – you need some appreciation of the ideas.

Derivatives, Tangents and Tangent Vectors

Figure 1 to the right shows the graph of a function, y(x), where y happens to be a cubic function of x. To be specific,

$$y = 5x^3 - 25x^2 + 35x - 7 \qquad \text{(CURV3-2)}$$

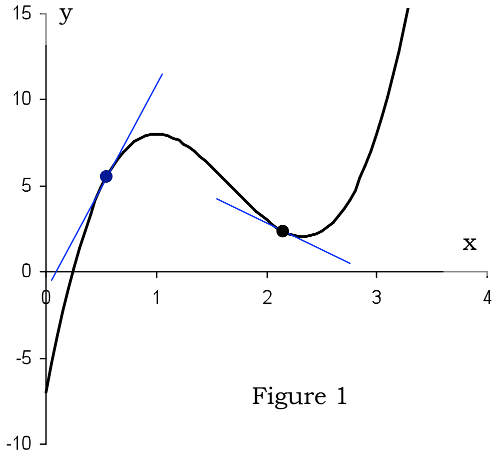


Figure 1

Because of the x^2 and x^3 terms in y, the graph is curved (rather than being a single straight line). Unlike a straight line, which has a constant slope throughout its entire extent, this curved graph appears to have a slope which varies in value (and sign) from one position or value of x to another. Further, sometimes the graph seems to be twisting in a clockwise direction (near x = 1) and in other places, it appears to be twisting in a counterclockwise direction (near x = 2), as viewed by someone moving along the curve from left to right. If we intend to be able to join small sections of curves together to form a smooth composite curve, we will need to ensure that not only do the individual curve segments meet, but that at those joints, they have something like the same direction, and perhaps even the same type of twist and hence, we need some sort of machinery to deal with variable slopes and "twists".

Differential calculus is the branch of mathematics that develops such machinery. A first step is to define what we mean by the slope of a curve as the slope of the straight line tangent to that curve at a particular point. A **tangent line** is a straight line that intersects a curve at a single point – just touching the curve at that point. Figure 1 shows two such lines, and you should be able to see that equating the slope of a curve at a point with the slope of the line tangent to that curve at a point seems to be reasonable.

Differential calculus then goes on to develop a set of recipes or formulas for calculating the slopes of tangent lines. You can see from Figure 1 that generally tangent lines at two different locations on the curve will have different slopes, and so what we expect is a formula for the slope of the tangent line which depends on the value of x, since the value of x tells us where we are on the curve. This formula for the slope of the tangent line as a function of x is called the **derivative** of y. In the case of the graph of y vs. x, this derivative is often denoted by the symbol y'(x) – spoken 'y-prime of x'. Another notation often used is $\frac{d}{dx}[y]$ *or* $\frac{dy}{dx}$, using the symbol $\frac{d}{dx}$ to denote the operation of finding the derivative of y when plotting against x. Although it looks a bit more complicated, this latter notation is a bit safer to use than the former because sometimes the context doesn't make clear exactly which variable is being plotted along the horizontal axis. The operator notation, d/dx, indicates clearly that we are giving a formula for the slope of the graph of y when y is plotted against x.

When we are working with simple polynomial functions, the recipe for obtaining a derivative turns out to be very simple. First, the derivative of a polynomial can be done term by term. Secondly, only the variable part of each term must be taken into account. The formula then says:

$$\frac{d}{dx}\left[Ax^n\right] = A\,nx^{n-1} \qquad \text{(CURV3-3)}$$

Thus, to find the derivative of polynomial, we just move through it, term-by-term, reducing the power of x in each term by 1, after first multiplying the term by the original power of x. For example, the derivative of the specific formula in equation (CURV3-2) is just

$$y = 5x^3 - 25x^2 + 35x - 7 \Rightarrow \qquad \frac{dy}{dx} = 5\cdot\left(3x^2\right) - 25\cdot\left(2x^1\right) + 35\cdot\left(1x^0\right) - 7\cdot(0)$$

or

$$\frac{dy}{dx} = 15x^2 - 50x + 35 \qquad \text{(CURV3-4)}$$

Notice that here we used the fact that $x^0 = 1$. Because any constant can be considered a term involving x^0, we end up with the additional result that the derivative of a constant is zero (which makes sense, because a derivative is a formula for a rate of change, and constants have zero rate of change!).

Tangent lines have been drawn at two specific points in Figure 1. The leftmost point has x = 0.55 (and substituting this into the formula (CURV3-2) gives y ≅ 5.52). If we substitute x = 0.55 into formula (CURV3-4), we get

$$\frac{dy}{dx} = 15\,(0.55)^2 - 50\,(0.55) + 35 = 12.0375$$

indicating that the slope of the tangent line at that point on the graph is 12.0375. This rather large positive value indicates a line pointing fairly steeply upwards to the right, which is what we see in the graph. The second point, to the right, occurs at x = 2.15, y ≅ 2.38. Substituting this value of x into our derivative formula gives

$$\frac{dy}{dx} = 15\,(2.15)^2 - 50\,(2.15) + 35 = -3.1625$$

This would be the slope of a line pointing at a much shallower angle upwards to the left, and again, that is precisely what we see in the figure.

You just need to recognize two additional ideas, and you've got all of the "calculus" that will be necessary to understand what follows in this document.

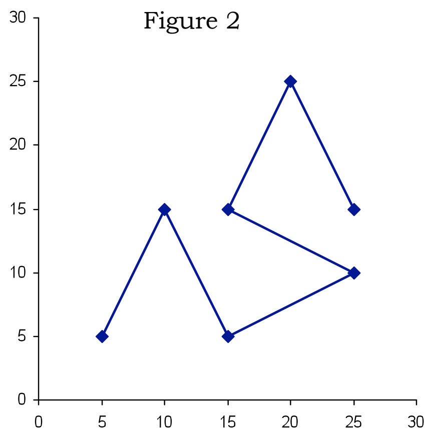
(i) First, the tangent line, and hence the derivative contains information about direction. If we were to represent the tangent direction as a line segment with an arrowhead, that arrow would be pointing in the direction of motion of an object moving along the curve. Thus, you can visualize the tangent line leading to the idea of a **tangent vector** associated with each point along a curve. When we move on to work with parametric equations for x and y, (and z, when you work in three dimensions), the components of this tangent vector will simply be the values of the derivatives if x, y, and z with respect to the parameter. As is true of all vectors, tangent vectors have both directions and magnitudes, and the fact that these two properties of a vector are independent is exploited in some of the methods we'll discuss below.

(ii) Secondly, since the derivative is itself a formula, it is possible to use the recipe described above to find the derivative of a derivative, more commonly called the **second derivative**, and denoted variously as y"(x) – spoken 'y double prime' – or $\frac{d^2y}{dx^2}$. This second derivative gives information on the direction and degree of "twist" in the graph at any point. In particular, when the second derivative has a negative value, the graph is twisting clockwise, and when the second derivative has a positive value, the graph is twisting counterclockwise. (Of course, a proper mathematician would never use such a trite word as "twist" in a discussion involving second derivatives! If you feel a burning desire to delve into this material more deeply, consult your nearest calculus textbook.)

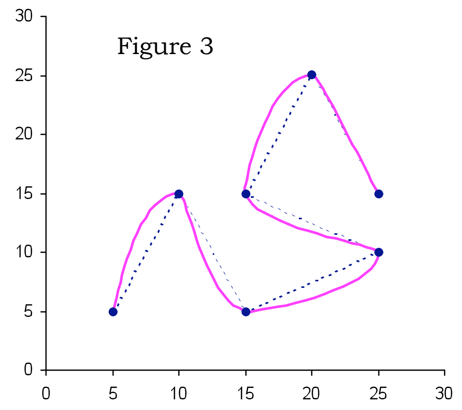
Degrees of Continuity

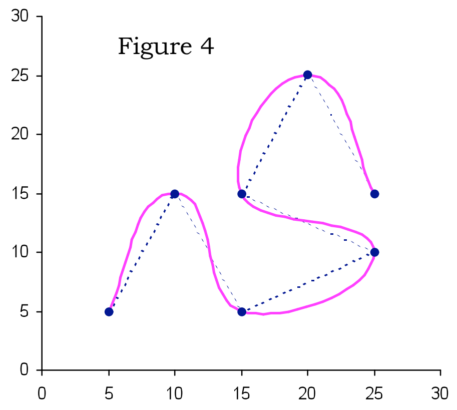
One more bit of notation or jargon before we get down to business with the curves. Mathematicians use the notation $\mathbf{C^k}$ and $\mathbf{G^k}$ to indicate, respectively, **continuity of order k** and **geometric continuity of order k** for a curve. As far as we're concerned here, this is an issue only at the joints between curve segments making up a composite curve.

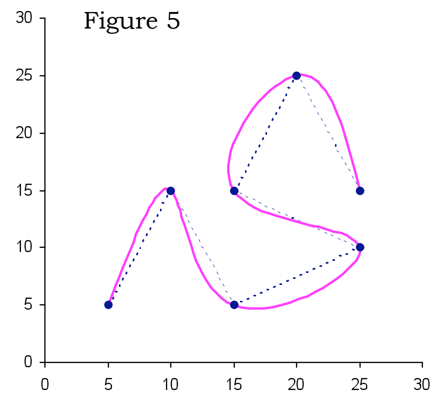
$\mathbf{C^0}$ and $\mathbf{G^0}$ essentially denote the same thing: the curve segments meet end to end, with nothing more implied about the smoothness of the joint. Thus, a polyline as shown in Figure 2 to the right has both $\mathbf{C^0}$ and $\mathbf{G^0}$ continuity. Generally, even if the pieces between the interpolation points are curves, a composite curve with this degree of continuity will have kinks as the joints.



$\mathbf{C^1}$ continuity occurs when the curve segments coming together at an interpolating point have identical tangent vectors at that point, both in direction and magnitude. For $\mathbf{G^1}$ continuity, the two segments must have tangent vectors with identical directions, but not necessarily identical magnitudes at the joint. Generally, larger magnitude tangent vectors tend to make the curve swing further away from the polyline through the interpolating points, so playing with the magnitudes of the tangent vectors can allow you to generate curves of greater or lesser "roundness." Or, if you like, imposing tangent vectors with smaller magnitudes tends to make the curve come away from a joint more closely along the line to the next joint. In Figure 3 below, there is just $\mathbf{G^1}$ continuity at each joint, with the tangent vectors on the outgoing side of the joint having just one-fifth magnitude of the tangent vectors on the inbound side of the joint (moving from left to right). Figure 4 shows the same situation but with the outbound tangent vectors given the same magnitude as the inbound tangent vectors at each point (so the curve in Figure 4 has $\mathbf{C^1}$ continuity). By boosting the magnitude of the tangent vectors even more, we can eventually cause the curve to describe loops in the vicinity of each joint.







As you'll see from the theory to be presented shortly, we have quite a bit of flexibility in how the $\mathbf{G}^1$ or $\mathbf{C}^1$ continuous composite curves are shaped.

With $\mathbf{C}^2$ continuity, virtually all of that flexibility is lost (though a little remains – see the discussion of clamped vs. free endpoints for natural cubic splines below). $\mathbf{C}^2$ continuity requires not only that the curve segments meet at the joints, have the same tangent vectors at the joints, but also, have the same second derivative vectors (remember the "twist"). For the set of points shown in Figures 2, 3, and 4, we get the curve in Figure 5 as an example of a $\mathbf{C}^2$ continuous curve using the "free endpoint" approach. Figure 5 appears at least marginally smoother and more flowing than Figure 4 – a result of requiring the second derivatives to match at the joints.

No commonly used methods attempt to achieve only $\mathbf{G}^2$ continuity by itself. One could go to a higher order continuity than the second, but this would require using curve segments which are fourth or higher order polynomials. Again, no commonly used methods do this. In fact, what makes methods based on cubic polynomial segments the most popular is that they are the simplest polynomials for which $\mathbf{C}^2$ continuity is possible, as well as being the simplest polynomials which can give non-planar curves (the two properties are not unrelated).

Natural Cubic Splines

"Natural Cubic Splines" are obtained by joining cubic polynomial segments to give a composite curve with $\mathbf{C}^2$ continuity. In this section, we will get the required formulas from an approach which explicitly forces the $\mathbf{C}^2$ continuity on the required algebraic form. Later on, essentially the same formulas will result when we force $\mathbf{C}^2$ continuity onto composite Hermite curves, giving an added insight into the characteristics of that approach.

Here, we will give quite a detailed derivation of the formulas to illustrate some of the mathematical ideas you need to take into account in developing many of the curve fitting formulas used in computer graphics. Of particular importance is making sure that indices are handled consistently so that all of the "unknowns" and "conditions" are properly taken into account.

Begin with a collection of L+1 control points: $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_L$, where the notation $\mathbf{p}_j$ stands collectively for the x-, y-, and if present, z-coordinates of the jth point. The idea here is to represent the curve between two successive points, $\mathbf{p}_j$ and $\mathbf{p}_{j+1}$ in this sequence by a parametric formula of the form:

$$\mathbf{p} = \mathbf{a}_j u^3 + \mathbf{b}_j u^2 + \mathbf{c}_j u + \mathbf{d}_j \qquad 0 \le u \le 1 \qquad \text{(CURV3-5)}$$

where boldface is used here for the characters **p, a, b, c,** and **d** to emphasize once again that there will be one equation of this form for each of the x, y, and z-coordinates along the curve. When u = 0, $\mathbf{p} \equiv \mathbf{p_j}$, and when u = 1, $\mathbf{p} \equiv \mathbf{p_{j+1}}$, in formula (CURV3-5).

Now, thinking of just one coordinate, either x, y, or z, at a time, we notice the following details about this approach. Since there are L+1 control points, there will be L intervals between control points. Figure 6 illustrates one such segment, which we call generically segment #j. Note that in our numbering scheme, the first point on the curve is point number 0, and the first segment in the composite curve is segment number 0. The formula (CURV3-5) for the curve between each pair of control points contains four constants, a_j, b_j, c_j, and d_j, to be determined, for a total of 4L such constants. This means we need to find 4L conditions that must be satisfied (which will turn into the 4L equations we need to solve to determine these 4L constants). These conditions arise as follows:

Figure 6

C^0 Continuity:

The formula (CURV3-5) must match the control points at each control point. Thus, taking u = 0 in (CURV3-5) for each interval, we get L conditions:

$$d_j = p_j\,, \qquad j = 0, 1, 2, \ldots L-1 \qquad \text{(CURV3-6a)}$$

and taking u = 1 for each interval gives another L conditions

$$a_j + b_j + c_j + d_j = p_{j+1}\,, \qquad j = 0, 1, 2, \ldots, L-1. \qquad \text{(CURV3-6b)}$$

This takes care of the $\mathbf{C}^0$ continuity and gives 2L of the 4L conditions required to determine the 4L unknowns in our model.

C^1 Continuity

The next step is to take into account the requirement for $\mathbf{C}^1$ continuity. For the moment, let the symbol $\mathbf{D}_j$ represent the tangent vector at point j, and we'll use the unbolded D_j to refer to a single component of that vector.

Now, the first derivative of formula (CURV3-5) is

$$p_j' = 3a_j u^2 + 2b_j u + c_j, \qquad j = 0, 1, 2, \ldots, L-1 \qquad \text{(CURV3-7)}$$

where, of course, p_j' stands for dp_j/du, the derivative of p_j with respect to the variable u. As seen from Figure 6 above, this formula must give D_j when u = 0, and D_{j+1} when u = 1:

$$c_j = D_j \qquad \text{(CURV3-6c)}$$
$$3a_j + 2b_j + c_j = D_{j+1} \qquad \text{(CURV3-6d)}$$

These two conditions exist for each one of the L segments, giving us what is apparently the other 2L conditions we require. And, indeed, if we ignore two very important problems temporarily (that we have surreptitiously included another L+1 unknowns in the problem – the D_j's; and that we haven't yet accounted for **C**^{2} continuity), we can actually solve the 4L equations given by (CURV3-6a, b, c, d) to get formulas for the unknown coefficients in the L formulas (CURV3-5).

In fact, look at just segment number j. From (CURV3-6a) and (CURV3-6c), we have

$$d_j = p_j \quad \text{and} \qquad c_j = D_j \qquad\qquad \text{(CURV3-8a)}$$

But the p_j's are coordinates of the interpolating points, and so we know their values, and we are for the moment pretending that we know what the values of the D_j's are (with the hope that a miracle will occur later on and make this true – well, not really.) Thus, two of the four coefficients in the formula for segment j are already known. This leaves a_j and b_j to be found. But, equations (CURV3-6b) and (CURV3-6d) are

$$a_j + b_j + c_j + d_j = p_{j+1}$$
$$3a_j + 2b_j + c_j = D_{j+1}$$

Substitute for c_j and d_j from equation (CURV3-8a), and move all the terms not containing a_j or b_j over to the right-hand side. This gives us

$$a_j + \quad b_j = p_{j+1} – p_j - D_j$$
$$3a_j + 2b_j = D_{j+1} - D_j$$

Don't let the detail on the right-hand sides worry you – each of those symbols just stands for numbers that we will plug in when solving an actual problem. Thus, this is really a simple set of two linear equations in two unknowns: a_j and b_j, and we can solve them in the same way we would solve any such equations. For example, multiply the first equation by 3, and then subtract from it the second equation. Only b_j survives on the left-hand side, and after a bit of collection of like terms on the right-hand side, we get

$$b_j = 3(p_{j+1} – p_j) – D_{j+1} – 2D_j \qquad\qquad \text{(CURV3-8b)}$$

Alternatively, we could multiply the first equation by –2 and add the result to the second equation. Then only a_j survives on the left-hand side, giving

$$a_j = -2(p_{j+1} – p_j) + D_j + D_{j+1} \qquad\qquad \text{(CURV3-8c)}$$

Formulas (CURV3-8a,b,c) give us formulas to calculate the four coefficients in the formula (CURV3-5) for every segment in our composite curve. The advantage of introducing the symbols D_j without attempting immediately to take into account the equations that will determine their values is that our original system of 4L equations in 4L unknowns turned out to be L identical and very simple systems of 4 equations in 4 unknowns which we could solve in general. Once we have a way of actually calculating values for the D_j's, we can use formulas (CURV3-8a, b, c) to calculate coordinates of points anywhere on the composite curve, and so render the curve to any desired precision.

C^2 Continuity

We still have L+1 unknowns: D_0, D_1, D_2, … D_L. We also still have to ensure that the second derivatives (the measures of "twist") of the curve segments match at the joints. This second derivative, p_j'', is obtained by taking the derivative of the "first" derivative given by (CURV3-7):

$$p_j'' = 6a_j u + 2b_j, \qquad j = 0, 1, 2, \ldots L-1 \qquad\qquad \text{(CURV3-9)}$$

At each joint, the values of these second derivatives must match. That is, the value of p_j'' at the right end of the jth segment must equal the value of p_{j+1}'' at the left end of the (j+1)th segment, as things are sketched in Figure 6. In symbols,

$$p_j'' (u = 1) = p_{j+1}'' (u = 0) \qquad\qquad \text{(CURV3-10a)}$$

and using (CURV3-9), this gives

$$6a_j + 2b_j = 2b_{j+1} \qquad \text{or} \qquad 3a_j + b_j = b_{j+1} \qquad \text{(CURV3-10b)}$$

You get one of these equations for each of j = 0, 1, 2, …, L-2 (since the last segment – segment number L-1 – does not have to match a further segment to its right). This is only L-1 conditions, and we have L+1 unknowns, so we're two conditions short.

There are two common ways to deal with this deficiency. Option 1, the **free-endpoint** case, is to recognize that if the curve is free to pivot on the end points, p_0 and p_L, then at those points the degree of "twist" should be zero. This gives two more equations:

$$p_0''(u = 0) = 2b_0 = 0 \qquad \text{or} \qquad b_0 = 0 \qquad \text{(CURV3-11a)}$$

and

$$p_L''(u = 1) = 6a_{L-1} + 2b_{L-1} \qquad \text{or} \qquad 3a_{L-1} + b_{L-1} = 0 \qquad \text{(CURV3-11b)}$$

Option 2, the **clamped endpoint** model, takes a somewhat easier way out. Since we have only enough equations to determine L-1 unknowns, we just let the user pick the values of the two extra unknowns. It is most useful to consider that the user selects desired values for D_0 and D_L, the directions of the composite curve at its two ends – this corresponds physically to "clamping" the curve to have a certain direction at each endpoint.

Now we have equations ((CURV3-10b) by themselves for the clamped endpoint model, and (CURV3-10b) combined with (CURV3-11a, b) for the free endpoint model). They need a bit of refinement before we can attempt a solution. Fortunately, a simple pattern emerges.

Looking at the free endpoint model, we take the equations in order from the first point to the last. At p_0, we have

$$b_0 = 0 \quad \Rightarrow \quad 3(p_1 - p_0) - D_1 - 2D_0 = 0$$

using (CURV3-8b), and since it is the D's that are unknown, we rearrange this to have the D-terms on the left hand side:

$$2D_0 + D_1 = 3(p_1 - p_0) \qquad \text{(CURV3-12a)}$$

Now, for points j = 1, 2, …, L-1, we have

$$3a_j + b_j = b_{j+1}$$

from (CURV3-10b), and substituting from (CURV3-8b, c) gives

$$3[-2(p_{j+1} - p_j) + D_j + D_{j+1}] + 3(p_{j+1} - p_j) - D_{j+1} - 2D_j = 3(p_{j+2} - p_{j+1}) - D_{j+2} - 2D_{j+1}$$

The right-hand side here is obtained by replacing j by j+1 everywhere in (CURV3-8b). It's messy, but not difficult in principle to gather all of the D's to the left side and everything else to the right side. This gives the much simpler looking equation

$$D_j + 4D_{j+1} + D_{j+2} = 3(p_{j+2} - p_j). \qquad \text{(CURV3-12b)}$$

Remember, there is one equation like this for every value of j from j = 1 to j = L-1. Finally, for point L, we have

$$3a_{L-1} + b_{L-1} = 0$$

which, by the same procedure, becomes

$$2D_L + D_{L-1} = 3(p_L - p_{L-1}) \qquad \text{(CURV3-12c)}$$

To see how "simple" the system of equations (CURV3-12a, b, c) really is, we'll write them out in a bit of detail:

$$
\begin{array}{lrcl}
j=0 & 2D_0 + D_1 & = & 3(p_1 - p_0) \\
j=1 & D_0 + 4D_1 + D_2 & = & 3(p_2 - p_0) \\
j=2 & D_1 + 4D_2 + D_3 & = & 3(p_3 - p_1) \\
j=3 & D_2 + 4D_3 + D_4 & = & 3(p_4 - p_2) \\
 & \ldots. & & \ldots. \\
 & D_j + 4D_{j+1} + D_{j+2} & = & 3(p_{j+2} - p_j) \\
 & \ldots. & & \ldots. \\
j=L-2 & D_{L-3} + 4D_{L-2} + D_{L-1} & = & 3(p_{L-1} - p_{L-2}) \\
j=L-1 & D_{L-2} + 4D_{L-1} + D_L & = & 3(p_L - p_{L-2}) \\
j=L & D_{L-1} + 2D_L & = & 3(p_L - p_{L-1})
\end{array}
$$

(CURV3-13)

Notice that there is a lot of simple pattern to these equations. Further, this is what is called a tridiagonal-band system of linear equations – if you picture a diagonal line running from the top left to the bottom right of the left-hand sides, you see that only the unknowns on this diagonal line and the single unknown to the immediate left and right of that one are present in each equation. It turns out that this it is possible to write down formulas for the solution of the system of equations directly (The details are spelled out in Appendix 1 of this document. Essentially, the first equation can be used to eliminate D_0 from the second. Then the new second equation can be used to eliminate D_1 from the third, and so on, until the new $(L-1)^{th}$ equation can be used to eliminate D_{L-1} from the last equation. The last equation then contains only one unknown, D_L, and so can be solved for D_L. But the second last equation now contains only D_{L-1} and D_L, and since D_L is now known, we can calculate D_{L-1}. Backtracking in this way up the list of equations gives us values for all of the unknowns. Again, see Appendix 1 for more details.)

The final solution is most easily stated as follows:

First, calculate the set of numbers in order:

$$\gamma_0 = \frac{1}{2}$$

$$\gamma_j = \frac{1}{4 - \gamma_{j-1}}, \quad j = 1, 2, \ldots, L-1 \qquad \text{(CURV3-14)}$$

$$\gamma_L = \frac{1}{2 - \gamma_{L-1}}$$

Now calculate in order

$$\delta_0 = (p_1 - p_0)\gamma_0$$

$$\delta_j = \left[3\left(p_{j+1} - p_{j-1}\right) - \delta_{j-1}\right]\gamma_j \quad , j = 1, 2, \ldots, L-1 \qquad \text{(CURV3-15)}$$

$$\delta_L = \left[3\left(p_L - p_{L-1}\right) - \delta_{L-1}\right]\gamma_L$$

Finally,.calculate in order

$$D_L = \delta_L$$

$$D_j = \delta_j - \gamma_j D_{j+1}, \quad j = L-1, L-2, \ldots., 1, 0 \qquad \text{(CURV3-16)}$$

Notice that here, we are calculating the D_j values in reverse order, from D_L down to D_0.

Remember, for each interval, from $\mathbf{p}_j$ to $\mathbf{p}_{j+1}$, the cubic formula can now be calculated using

$$a_j = 2(p_j - p_{j+1}) + D_{j+1} + D_j$$
$$b_j = 3(p_{j+1} - p_j) - D_{j+1} - 2D_j$$
$$c_j = D_j$$
$$d_j = p_j \qquad \text{(CURV3-17)}$$

The calculations detailed in (CURV3-14) – (CURV3-17) must be done for each coordinate: x, y, and if present, z. From a computer programming viewpoint, this just means that all of the symbols on the left-hand sides of these formulas will show up in program code as two or three column arrays. In the absence of some very careful and sophisticated programming technique, these computations should be done using at least single precision floating point arithmetic.

Example

As an example calculation, we determine the free endpoint natural cubic spline curve for the eight control points example used in earlier documents in this series (see Figure 4 in CURVES_I_BEZIER.DOC, for example). First, we need to give the actual coordinates of the eight control points:

	x	**y**
p_0	0	2
p_1	1	5
p_2	2.5	3.4
p_3	3	2
p_4	4	2.5
p_5	5	4
p_6	6	5
p_7	8	1

Since we are dealing with a two-dimensional figure, each point requires only x- and y-coordinates. In this case, there are eight control points, numbered from 0 to 7, hence L = 7 in our formulas.

We must do the calculations indicated by formulas (CURV3-14), (CURV3-15) and (CURV3-16) in order first. These have been set up in the Excel worksheet in file NATCUB8.XLS. The results of these three sets of equations are summarized in the table:

j	γ_j	$\delta_{j,x}$	$D_{j,x}$	$\delta_{j,y}$	$D_{j,y}$
0	0.50000	1.50000	0.78495	4.50000	4.21450
1	0.28571	1.71429	1.43009	-0.08571	0.57101
2	0.26923	1.15385	0.99468	-2.40000	-2.29852
3	0.26804	0.89691	0.59121	-0.08041	-0.37692
4	0.26796	1.36740	1.14050	1.62928	1.10618
5	0.26795	1.24130	0.84679	1.57306	1.95218
6	0.26795	2.07894	1.47235	-2.83304	-1.41491
7	0.57735	2.26383	2.26383	-5.29255	-5.29255
	(CURV3-14)	(CURV3-15) for x	(CURV3-16) for x	(CURV3-15) for y	(CURV3-16) for y

We haven't given any details of the arithmetic here, but you should be able to use equations (CURV3-14) through (CURV3-16) to duplicate the results without much trouble. The columns based on (CURV3-14) and (CURV3-15) are calculated from top to bottom, those based on (CURV3-16) are calculated from bottom to top. In calculating the δ_j for x, use the x-coordinates of the control points, and, of course, to get the δ_j for y, use the y-coordinates of the control points.

With the information contained in the table above, it is now possible to use equations (CURV3-17) to work out the coefficients of the cubic curves giving each segment of the overall curve. As an example, we do this for the third segment, running from the point $p_2 = (2.5, 3.4)$ to $p_3 = (3, 2)$. For the x-coordinates, we get

$$d_2 = (p_2)_x = 2.5$$
$$c_2 = (D_2)_x = 0.99468$$
$$b_2 = 3[(p_3)_x - (p_2)_x] - (D_3)_x - 2(D_2)_x = 3[3 - 2.5] - 0.59121 - 2(0.99468) = -1.08056$$
$$a_2 = 2[(p_2)_x - (p_3)_x] + (D_3)_x + (D_2)_x = 2[2.5 - 3] + 0.59121 + 0.99468 = 0.585881$$

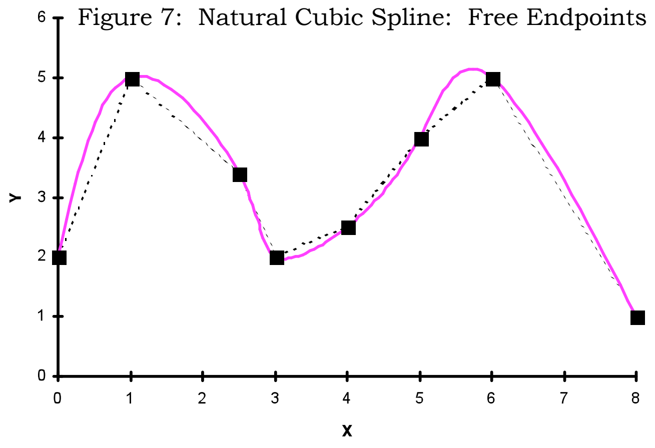
Here all calculations have been rounded to the indicated number of decimal places. In the same way, we get for the y-coordinates,

$$d_2 = 3.4, \quad c_2 = -2.29852, \quad b_2 = 0.773961, \quad a_2 = 0.124562$$

remembering that here, we work with the y-coordinates of the two control points involved. This means that the cubic curve going from p_2 to p_3 will have the parametric equations:

$$x = 2.5 + 0.99468u - 1.08056u^2 + 0.585881u^3$$
$$y = 3.4 - 2.29852u + 0.773961u^2 + 0.124562u^3$$

By calculating the (x, y)-coordinates using a series of values of u running from u = 0 (point p_2) to u = 1 (point p_3), and joining them in succession by straight line segments, the curve in Figure 7 results. To produce the curve in Figure 7, we used 20 values of u for each line segment: u = 0, 0.05, 0.10, 0.15, ... 0.95 and 1. (One caution: these spreadsheet examples have evolved over time and various versions of the files are floating around – some may have the order of the symbols a, b, c, and d in formula (CURV3-1) or (CURV3-5) reversed, and may also refer to the parameter as t rather than u.)



You see from this example that the resultant 7-piece curve runs relatively smoothly through all eight control points (that is, it **interpolates** the control points). The twin-hump shape of the curve is very pronounced as a result, compared with the shape of the eight-point Bezier curve. Except for a bit of an awkward swing between p_5 and p_6 (the second last pair of control points), this natural cubic spline follows the underlying polyline quite closely. At each control point, the segments of the curve not only meet end-to-end (C^0-continuity), but they have the same direction (C^1-continuity -- no kinks at the control points), and they are "twisting" in the same sense (C^2-continuity). To see this last property, notice that as the curve passes through the second control point, p_1, both the incoming segment and the outgoing segment are twisting clockwise. At the fourth control point, both incoming segment and outgoing segment are twisting counterclockwise.

◆◆◆

That may have seemed like a lot of work, but the result is quite remarkable: we've come up with explicit formulas for cubic polynomial segments that join together smoothly over an arbitrarily long sequence of interpolating points. Part of the seeming complexity of the work above is due to our goal of accurately accounting for all indices and variables. This is, of course, mandatory in any endeavor relating to computer programming, because the infamous "off-by-one" error has brought many a program to a crashing halt.

Fortunately, we don't have to repeat all of this work to get the formulas for the clamped endpoint natural cubic spline. Instead, we just note that the clamped endpoint model leads to a system of equations that is just (CURV3-13), but with the j = 0 and j = L rows (and their corresponding columns) deleted. The right-hand side of the j = 1 equation picks up a term '$-D_0$', and the right-hand side of the j = L-1 equation picks up the term '$-D_L$'. The new system is an even simpler tridiagonal band system. The solution is very similar to (CURV3-14) – (CURV3-16.

First calculate in order

$$\gamma_1 = \frac{1}{4}$$

$$\gamma_j = \frac{1}{4-\gamma_{j-1}}, \; j = 2, \ldots, L\text{-}2 \qquad \text{(CURV3-18)}$$

Now calculate in order

$$\delta_1 = \left[3(p_2 - p_0) - D_0\right]\gamma_1$$
$$\delta_j = \left[3\left(p_{j+1} - p_{j-1}\right) - \delta_{j-1}\right]\gamma_j \qquad , j = 2, \ldots, L\text{-}2 \qquad \text{(CURV3-19)}$$
$$\delta_{L-1} = \left[3\left(p_L - p_{L-2}\right) - D_L - \delta_{L-2}\right]\gamma_{L-1}$$

Finally,.calculate in order

$$D_{L-1} = \delta_{L-1}$$
$$D_j = \delta_j - \gamma_j D_{j+1}, \quad j = L\text{-}2, \ldots 2, 1 \qquad \text{(CURV3-20)}$$

The application of these formulas is very similar to the application of the free endpoint model formulas.

Example

In the free endpoint example presented just above, we found that the tangent vectors at the first and last points turned out to be:

	$D_{j,x}$	**$D_{j,y}$**	**Comments**
j = 0	0.78495	4.21450	pointing steeply upwards to right
j = 7	2.26383	-5.29255	pointing moderately downwards to right

Now, using the clamped endpoint model, let's pre-set these two tangent vectors as follows:

	$D_{j,x}$	**$D_{j,y}$**	**Comments**
j = 0	-4	4	pointing moderately downwards to the right
j = 7	2	5	pointing moderately upwards to right

All the details are laid out in the file NATCUB8.XLS. In summary, formulas (CURV3-18) through (CURV3-20) give

j	γ_j	$\delta_{j,x}$	$D_{j,x}$	$\delta_{j,y}$	$D_{j,y}$
0	not req'd	not req'd	-4.00000	not req'd	4.00000
1	0.25000	2.87500	2.71213	0.05000	0.63202
2	0.26667	0.83333	0.65149	-2.41333	-2.32807
3	0.26786	0.98214	0.68190	-0.07679	-0.31975
4	0.26794	1.34450	1.12092	1.62823	0.90708
5	0.26795	1.24744	0.83442	1.57333	2.69145
6	0.26795	1.54139	1.54139	-4.17286	-4.17286
7	not req'd	not req'd	2.00000	not req'd	5.00000
	(CURV3-18)	(CURV3-19) for x	(CURV3-20) for x	(CURV3-19) for y	(CURV3-20) for y

The shaded cells in this table are the D_j's that have been fixed in advance. The effect of these values should be to make the composite curve approach the initial point on the left end moving in a downwards to the right direction, and the final point on the right moving in an upwards to the right direction. In both cases, these directions are not the directions of approach to these points if one were following the polyline through the interpolating points, so we expect there will be some change in the shape of the curve to accommodate these "demands.". Figure 8 shows that this is indeed what happens.

◆◆◆

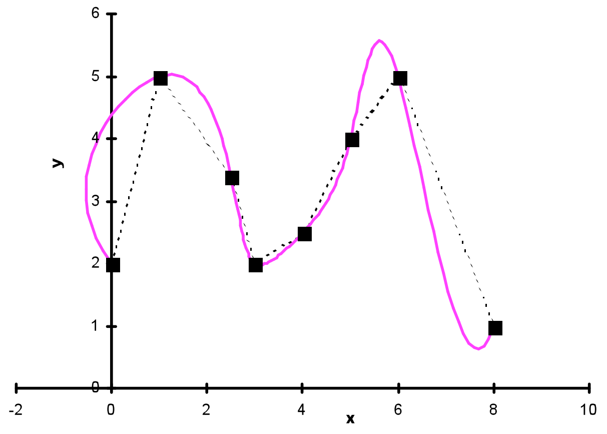


Figure 8: Natural Cubic Spline with Clamped Endpoints

The Hermite Curve

The Hermite curve is a very popular starting point for many applications, and for the development of curve generation methods based on cubic polynomials, but having special properties. In actual fact, the Hermite approach doesn't really introduce much in the way of new concepts over the natural cubic spline approach we've just covered in gruesome detail. However, it does provide a somewhat different focus on how the various bits come together to produce a particular curved shape. Also, it is more natural to introduce a matrix form here, and this will make it easier to inter-relate the geometric information required by various cubic spline methods.

The Hermite approach starts with a cubic polynomial segment given by formula (CURV3-1), but is based on the provision of the coordinates of the two endpoints of the segment (p_0 and p_1), and the tangent vectors at the two endpoints (D_0 and D_1), as sketched in Figure 9 to the right. For the moment, we are just dealing with one segment, and so it isn't necessary to place subscripts on the various symbols. Writing the cubic polynomial as

Figure 9

$$p(u) = au^3 + bu^2 + cu + d$$

where $0 \le u \le 1$ generates the segment from p_0 to p_1, we then also have

$$p'(u) = 3au^2 + 2bu + c.$$

Thus, as we found early in our development of the natural cubic spline formulas, we can write:

$$p_0 = p(u = 0) = d$$

$$p_1 = p(u = 1) = a + b + c + d$$

$$D_0 = p'(u = 0) = c$$

and

$$D_1 = p'(u = 1) = 3a + 2b + c$$

These four equations are easily solved for the four coefficients a, b, c, and d, giving

$a = -2(p_1 - p_0) + D_0 + D_1$
$b = 3(p_1 - p_0) - 2D_0 - D_1$
$c = D_0$
$d = p_0$ (CURV3-21)

This result should look familiar – it's formulas (CURV3-8a, b, c) specialized to a single cubic polynomial segment. Now, we can substitute these expressions back into the formula for p(u), obtaining

$$p(u) = [-2(p_1 - p_0) + D_0 + D_1]\, u^3 + [3(p_1 - p_0) - 2D_0 - D_1]\, u^2 + D_0 u + p_0$$

or, rearranging this right-hand side so that the terms are grouped as coefficients of p_0, p_1, D_0, and D_1, we get

$$p(u) = (2u^3 - 3u^2 + 1)\, p_0 + (-2u^3 + 3u^2)\, p_1 + (u^3 - 2u^2 + u)\, D_0 + (u^3 - u^2)\, D_1 \qquad \text{(CURV3-22)}$$

Here, the functions

$F_1(u) = 2u^3 - 3u^2 + 1$
$F_2(u) = -2u^3 + 3u^2$
$F_3(u) = u^3 - 2u^2 + u$
$F_4(u) = u^3 - u^2$ (CURV3-23)

are the so-called **Hermite Basis Functions**, and play the role here of blending functions – their values measure how much influence each of the four items of geometric information has on the shape of the curve at any given value of u.

It is common to write formula (CURV3-22) in a matrix form. The easiest way to see this develop is to note first of all that formula (CURV3-1) itself can be written as a matrix product:

$$p = au^3 + bu^2 + cu + d = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \bullet \begin{bmatrix} d \\ c \\ b \\ a \end{bmatrix} \qquad \text{(CURV3-24a)}$$

or

$$\mathbf{p} = \mathbf{U} \cdot \mathbf{A} \qquad \text{(CURV3-24b)}$$

where we've used boldface in this second form to indicate that these quantities are really matrices themselves. **p** stands for the matrix whose rows contain the coordinates of points on the curve (each row having 2, 3, or 4 elements, this last value if one is working with homogeneous coordinates in three dimensions). The matrix **U** is a matrix with four-element rows, each row containing the first four powers of a value of u. Finally, **A** is the matrix with columns containing the coefficients a, b, c, and d in (CURV3-1), one column for each coordinate direction in use. Formula (CURV3-24) expresses the equation of the cubic polynomial segment in terms of the algebraic coefficients of the polynomial, contained in matrix **A**.

But, we can also write

$$p = F_1\, p_o + F_2\, p_1 + F_3\, D_0 + F_4\, D_1 = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 \end{bmatrix} \bullet \begin{bmatrix} p_0 \\ p_1 \\ D_0 \\ D_1 \end{bmatrix} = \mathbf{F} \cdot \mathbf{B} \qquad \text{(CURV3-25a)}$$

and further

$$\mathbf{F} = \begin{bmatrix} 2u^3 - 3u^2 + 1, & -2u^3 + 3u^2, & u^3 - 2u^2 + u, & u^3 - u^2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} = \mathbf{U} \cdot \mathbf{M_F} \qquad \text{(CURV3-25b)}$$

so that the equation for the cubic polynomial can also be written as

$$\mathbf{p} = \mathbf{U} \cdot \mathbf{M_F} \cdot \mathbf{B} \qquad \text{(CURV3-26)}$$

This formula expresses the equation of the cubic polynomial segment in terms of the geometric information contained in the matrix **B**. The matrix, $\mathbf{M_F}$ here is the so-called 'Hermite basis matrix' and contains the information necessary to turn the geometric information in **B** into the desired cubic polynomial. Furthermore, since it's the same polynomial in either case,

$$\mathbf{p} = \mathbf{U} \cdot \mathbf{A} = \mathbf{U} \cdot \mathbf{M_F} \cdot \mathbf{B} \qquad \text{(CURV3-27a)}$$

we are justified in writing (by comparing the second and third parts above)

$$\mathbf{A} = \mathbf{M_F} \cdot \mathbf{B} \qquad \text{(CURV3-27b)}$$

from which we can write

$$\mathbf{B} = \mathbf{M_F}^{-1} \cdot \mathbf{A} \qquad \text{(CURV3-27c)}$$

where the inverse matrix $\mathbf{M_F}^{-1}$ can be determined to be

$$\mathbf{M_F}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \qquad \text{(CURV3-28)}$$

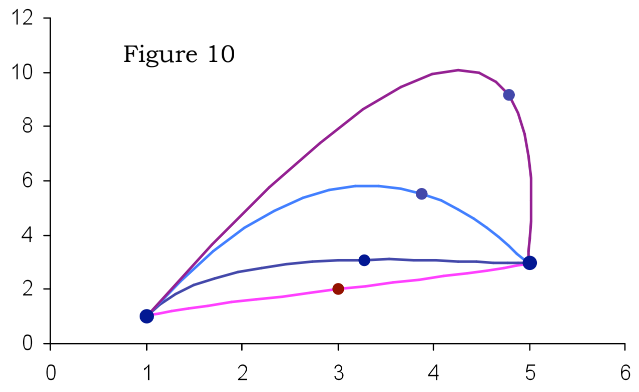
Formulas (CURV3-27b, c) are very useful, because they provide an easy way to switch between the algebraic specification of a curve, and its corresponding geometric specification. Further, practitioners find matrix formulations of this sort very useful, because they are easy to program, and they make relationships between different approaches quite transparent. The recipes for many of the curve generation methods used in computer graphics can be represented as the product of three matrices: a matrix of parameter powers (**U** above); a basis matrix specific to the method under consideration but not dependent in any way on the geometric shape information (for example, $\mathbf{M_F}$ above); and a situation specific matrix of shape information (for example, **A** or **B** above). When different representations of the same curve are given in this general format, it is easy to set up a program to accept information in any particular form preferred by the user, and convert it to other forms as required later.

If this last bit of work seems a bit mysterious, perhaps do a quick re-read of the document MATRIX.DOC.

Just before moving on to a short discussion of composite Hermite curves, we note that it is often useful to write the two tangent vectors in the form

$$\mathbf{D}_0 = k_0\, \mathbf{t}_0 \qquad \text{and} \qquad \mathbf{D}_1 = k_1\, \mathbf{t}_1 \qquad \text{(CURV3-29)}$$

where $\mathbf{t}_0$ and $\mathbf{t}_1$ are unit vectors in the direction of the desired tangent vector, and k_0 and k_1 are constants, giving the magnitudes of $\mathbf{D}_0$ and $\mathbf{D}_1$, respectively. Figure 10 shows the effects of varying the relative values of k_0 and k_1 while keeping the directions of the tangent vectors unchanged. This gives a degree of control over the shape of the curve that is relatively easy to exploit compared with trying to change both the magnitude and direction of the tangent vectors together.



In Figure 10, p_0 is the control point on the left, p_1 the control point on the right. $\mathbf{t}_0$ points steeply up to the right, and $\mathbf{t}_1$ points steeply down to the right. k_1 is fixed at 1, and k_0 varies through the series 1, 10, 30, 60, as you move from the lowest, straightest curve to the top curve. The point drawn in the interior of each curve corresponds to $u = 0.5$ in each case. The effect of increasing the magnitude of one tangent vector compared to the other is readily seen. If the magnitudes of both tangent vectors are increased simultaneously, the curve gets a more and more pronounced twist. You can use the simple Excel workbook called HERMITEG.XLS to experiment with this effect further.

<u>Composite Hermite Curves</u>

Joining cubic Hermite curve segments together to achieve either $\mathbf{C}^0/\mathbf{G}^0$, $\mathbf{C}^1$, $\mathbf{G}^1$, or $\mathbf{C}^2$ continuity is almost trivially easy, given their definition as illustrated in Figure 9 and the work we've already done in connection with natural cubic splines.

Obviously, to ensure that either $\mathbf{C}^0/\mathbf{G}^0$ occurs, we just have to ensure that $\mathbf{p}_0$ for one segment is identical to $\mathbf{p}_1$ for the previous segment. No other restrictions need be imposed.

For $\mathbf{G}^1$ continuity, we need to ensure that $\mathbf{D}_0$ for one segment is a constant multiple of $\mathbf{D}_1$ for the previous segment. $\mathbf{C}^1$ continuity requires that $\mathbf{D}_0$ of one segment is exactly equal to $\mathbf{D}_1$ of the preceding segment. In fact, if we number the interpolating points of the composite curve with subscripts beginning with $j = 0$ (so that the composite curve will be determined by the sequence of points $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, ... with the corresponding tangent vectors $\mathbf{D}_0$, $\mathbf{D}_1$, $\mathbf{D}_2$, ...) and then number the individual cubic Hermite segments starting with the index $j = 0$, (so that the index of a cubic Hermite segment is the same as the index on its beginning interpolating point), we will have that $\mathbf{C}^1$ continuity will result if the tangent vector at the end of segment j is $\mathbf{D}_{j+1}$. But this is just formula (CURV3-6d) seen earlier, and so, simply arrive back at formulas (CURV3-8a, b, c).

Going further to require $\mathbf{C}^2$ continuity will now just take us back through the sequence of equations beginning with (CURV3-9) in our discussion of natural cubic splines. Thus, joining cubic Hermite polynomial segments together to form a composite curve with $\mathbf{C}^2$ continuity just gives us the natural cubic spline examined in so much detail earlier on.

The value that the cubic Hermite polynomial adds to this overall discussion is its focus on the tangent vectors at the interpolating points. Where , $\mathbf{C}^1$ or $\mathbf{G}^1$ continuity as an end is not handled very intuitively in the development of the natural cubic spline formulas, it's implementation is much more obvious when starting with the Hermite approach.

Cardinal Splines

Formula (CURV3-27a) expresses the coordinates of points along a cubic Hermite polynomial segment in a simple matrix form. This is a very attractive form to work with because of the ease of manipulation of such matrix equations. Unfortunately, when one attempts to create a composite curve with more than $\mathbf{C}^0$ continuity, the geometric information specifying successive segments of the composite curve gets mixed together in a less intuitive fashion, and a segment-by-segment matrix formula of the sort exemplified by (CURV3-27a) is no longer possible or feasible. The problem here is really that of non-locality – once you impose $\mathbf{C}^1$ or $\mathbf{C}^2$ continuity on a composite curve, then changing a single piece of geometric information (the coordinates of just one interpolating point, or the tangent vector at one such point) is enough to cause the shape of the entire composite curve to change.

One approach to solving this problem has been to try to define the tangent vector at an interpolating point in terms of the coordinates of that point and nearby interpolating points. You sacrifice some freedom or detailed control over the shape of the curve, but for many applications, this loss of control is considered amply compensated by the much simpler formulas that result. (The perfectionist will bemoan not being able to specify his/her own tangent vectors, but the insouciant will see this as a chance to be relieved of the responsibility to provide tangent vectors!)

The so-called **cardinal splines** result from expressing the tangent vector at point $\mathbf{p}_j$ in terms of the coordinates of immediately preceding and following points. The simple formula

$$\mathbf{D}_j = \frac{1}{2}\left(\mathbf{p}_{j+1} - \mathbf{p}_{j-1}\right) \qquad \text{(CURV3-30)}$$

leads to the so-called **Catmull-Rom splines**. However, many practitioners introduce an additional "tension" parameter t, as well:

$$\mathbf{D}_j = \frac{1}{2}(1-t)\left(\mathbf{p}_{j+1} - \mathbf{p}_{j-1}\right) = s\left(\mathbf{p}_{j+1} - \mathbf{p}_{j-1}\right) \qquad \text{(CURV3-31)}$$

Here, $s = (1 - t)/2$. Negative values of t (or $s > ½$) give broader, rounder curves, whereas positive values of t ($s < ½$) give sharper, more angular curves that tend to follow the polyline through the interpolating points more closely. Obviously, (CURV3-30) is the special case of (CURV3-31) with $t = 0$ or $s = ½$. In either case, this approach generates a tangent vector at point j which is parallel to the direction joining points j-1 and j+1. The tension parameter simply affects the magnitude of the tangent vector used, but not its direction. The resulting composite curves (now specified simply by the set of interpolating points) will thus have $\mathbf{C}^1$ continuity (there is just one tangent vector defined at each interpolating point, associated with both segments meeting at that point), but in general will not have $\mathbf{C}^2$ continuity.

Now, substituting (CURV3-31) into formulas (CURV3-8a, b, c) gives

$$d_j = p_j$$

$$c_j = D_j = s(p_{j+1} - p_{j-1}) = sp_{j+1} - sp_{j-1}$$

$$\begin{aligned} b_j &= 3(p_{j+1} - p_j) - D_{j+1} - 2D_j \\ &= 3(p_{j+1} - p_j) - s(p_{j+2} - p_j) - 2\,s(p_{j+1} - p_{j-1}) \\ &= 3p_{j+1} - 3p_j - sp_{j+2} + sp_j - 2sp_{j+1} + 2sp_{j-1} \\ &= -sp_{j+2} + (3-2s)p_{j+1} - (3-s)p_j + 2sp_{j-1} \end{aligned}$$

and finally,

$$\begin{aligned} a_j &= -2(p_{j+1} - p_j) + D_j + D_{j+1} \\ &= sp_{j+2} + (s-2)p_{j+1} + (2-s)p_j - sp_{j-1} \end{aligned} \qquad \text{(CURV3-32)}$$

Thus, the coefficients in the cubic polynomial formula for segment number j depend on the coordinates of the four points p_{j-1}, p_j, p_{j+1} and p_{j+2}. In fact, substituting (CURV3-32) into the basic cubic polynomial formula (CURV3-1), and grouping terms involving point coordinates gives

$$P_j(u) = [su^3 - su^2]\, p_{j+2} + [\,(s-2)u^3 + (3-2s)u^2 + su]\, p_{j+1} + [\,(2-s)u^3 + (s-3)u^2 + 1]\, p_j + [-su^3 + 2su^2 - su]\, p_{j-1} \qquad \text{(CURV3-33a)}$$

or, in matrix form

$$\mathbf{p_j}(u) = \mathbf{U}\ \mathbf{M_{CS}}\ \mathbf{P_j}$$

$$= \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -s & 0 & s & 0 \\ 2s & s-3 & 3-2s & -s \\ -s & 2-s & s-2 & s \end{bmatrix} \bullet \begin{bmatrix} p_{j-1} \\ p_j \\ p_{j+1} \\ p_{j+2} \end{bmatrix} \qquad \text{(CURV3-33b)}$$

Symbols get a bit overworked here. The $\mathbf{p}_j(u)$ on the left-hand side stands for points on the jth segment of the generated curve. The matrix $\mathbf{P}_j$ on the right hand side stands for the matrix of coordinates of the four interpolating points determining segment j of the curve. The segment generated by (CURV3-33) will pass through points p_j (corresponds to u = 0) and p_{j+1} (corresponds to u = 1).

Formula (CURV3-33) requires an apparently nonexistent point p_{-1} to construct the segment j = 0, and it requires another apparently nonexistent point p_{L+1} to construct the final segment, for j = L-1. There are at least two common ways to deal with this problem. One is to simply not draw the j = 0 and the j = L-1 segments of the composite curve, using the initial and final specified points, p_0 and p_L as necessary information for the j = 1 and j = L-1 segments, respectively. These two points might appear on screen during the phase where the curve is being designed, but afterwards, both act as usually not visible control points. This means that the composite curve that results really runs from p_1 to p_{L-1}. This is the method illustrated in Foley, Van Dam, et al, Figure 11.32, page 505.

Others use a way of generating the necessary p_{-1} and p_{L+1} in some reasonable fashion. A common way to do this is to locate p_{-1} on the line between p_0 and p_1 at a distance of $|\ \mathbf{p}_1 - \mathbf{p}_0\ |$ from p_o (see Figure 11 to the right). Using vector arithmetic, we get

$$\begin{aligned} \mathbf{p}_{-1} &= \mathbf{p}_0 - (\mathbf{p}_1 - \mathbf{P}_0) \\ &= 2\mathbf{p}_0 - \mathbf{p}_1 \qquad \text{(CURV3-34a)} \end{aligned}$$

Similarly, p_{L+1} is located on the line joining $p_{L-1}$and p_L at a distance $|\ \mathbf{p}_L - \mathbf{p}_{L-1}\ |$ beyond p_L. Again, using vector arithmetic, we get

Figure 11

$$\begin{aligned} \mathbf{P}_{L+1} &= \mathbf{p}_L + (\mathbf{p}_L - \mathbf{p}_{L-1}) \\ &= 2\mathbf{p}_L - \mathbf{p}_{L-1} \qquad \text{(CURV3-34b)} \end{aligned}$$

Using these formulas allows you to extend the composite curve to include both p_0 and p_L.

Example
As an example, we'll briefly set up the Cardinal Spline curve corresponding to the set of interpolating points used earlier in this document in Figures 2 through 5. There are seven control points in the original example, and we will extend that list by two, using formulas (CURV3-34). This leaves us with the set of coordinates:

	x	y
p_{-1}	0	-5
P_0	5	5
P_1	10	15
P_2	15	5
P_3	25	10
P_4	15	15
P_5	20	25
P_6	25	15
P_7	30	5

The shaded boxes represent the coordinates computed using equations (CURV3-34). For p_{-1}, we get

$$x = 2(p_0)_x - (p_1)_x = (2)(5) - 10 = 0$$

and
$$y = 2(p_0)_y - (p_1)_y = (2)(5) - 15 = -5$$

whereas for p_7, we get

$$x = 2(p_6)_x - (p_5)_x = (2)(25) - 20 = 30$$

and
$$y = 2(p_6)_y - (p_5)_y = (2)(15) - 25 = 5$$

Now, when the tension parameter t = 0, or s = ½, the "basis" matrix, $\mathbf{M}_{CS}$ becomes from (CURV3-33b):

$$M_{CS} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.5 & 0 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 1.5 & -1.5 & 0.5 \end{bmatrix}$$

Now, for the segment of the curve from p_0 to p_1, the matrix $\mathbf{P}_0$ on the extreme right of (CURV3-33) is just the first four rows of the coordinate matrix above. Thus, the equation for the cubic polynomial giving this first segment of the overall curve is

$$p_0(u) = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.5 & 0 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 1.5 & -1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0 & -5 \\ 5 & 5 \\ 10 & 15 \\ 15 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 5 & 5 \\ 5 & 10 \\ 0 & 10 \\ 0 & -10 \end{bmatrix} = \begin{bmatrix} 5 + 5u, & 5 + 10u + 10u^2 - 10u^3 \end{bmatrix}$$

The two components of this final matrix are just the parametric formulas for the x and y coordinates of points along the cubic cardinal spline curve between points p_0 and p_1. By starting at the point $p_0 = (5, 5)$, and executing a sequence of LineTo() operations to points with coordinates resulting from substituting, say, u = 0.05, 0.10, 0.15, ..., 0.95, and 1, you'll get an approximation to a smooth curve between these first two interpolating points. The parametric equations for the (x, y) coordinates of points on the curve segment between p_1 and p_2 are obtained in the same way, but with the 4 x 2 matrix on the extreme right of the first line above replaced by one consisting of the second through fifth rows of the coordinates table above (the rows labelled p_0, p_1, p_2, and p_3). Repeat this process six times in total to get the six curve segments forming the composite curve between p_0 and p_6, and passing through the five points in between.

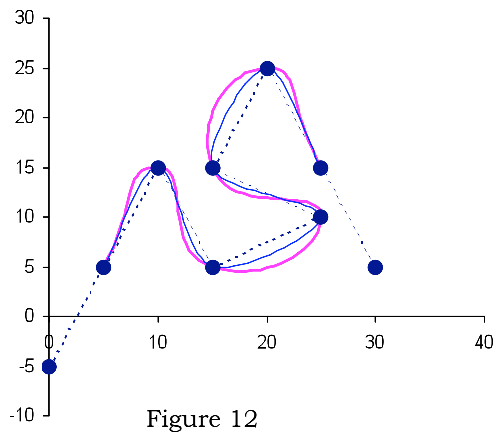


Figure 12

Figure 12 to the right shows the end result. The heavy dots are the interpolating points, including the two generated using formulas (CURV3-34) to start the process off. For reference, these points are joined by a dotted line showing the polyline through the interpolating points themselves. The lighter-weight curve generally closer to this polyline is the curve that results when the tension parameter t is zero. The composite curve is quite smooth, but doesn't stray too far from the control polyline in this case. The other heavier-weight curve resulted from setting t = -1. The increased roundness (or decreased tightness, if you like) is readily obvious. Using t = 1 gives a curve which is visually indistinguishable from the polyline at this scale. Using much larger negative values of t leads to composite curves with strange kinks and loops, and so forth. This figure is set up in the Excel workbook KBSCURVE.XLS, and you can use that workbook experiment with the value of t in greater detail if you wish. *[Perhaps read the next section on the Kochanek-Bartels spline before attempting to play with KBSCURVE.XLS.]*

<u>The Kochanek-Bartels Spline</u>

We just mention one more variation on the cardinal spline before moving on to our final major variant of interpolating cubic spline problem. The cardinal spline model imposes **C**^{1} continuity via formulas (CURV3-30) or (CURV3-31). In 1984, Kochanek and Bartels published a somewhat complicated replacement formula for (CURV3-31) which includes two additional parameters, b and c, that give you a way to systematically destroy the first order continuity of the cardinal spline in a way that's particular useful in generating curves to function as animation paths.

The Kochanek-Bartels approach allows for the tangent vector inbound to an interpolating point to be different than the tangent vector outbound. At interpolating point j, their formulas give:

$$D_j^{out} = s[\ (1+b)(1+c)(p_j - p_{j-1}) + (1-b)(1-c)(p_{j+1} - p_j)\ ]$$

and (CURV3-35)

$$D_j^{\,in} = s[\ (1+b)(1-c)(p_j - p_{j-1}) + (1-b)(1+c)(p_{j+1} - p_j)\ ]$$

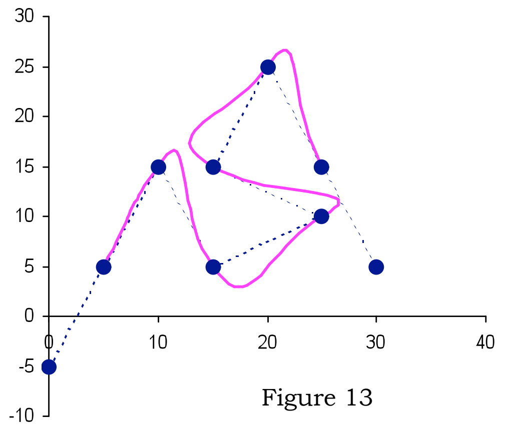


Figure 13

Here, $D_j^{\,in}$ stands for the tangent vector at the end of segment j-1, coming "in" to point number j, and D_j^{out} stands for the tangent vector at point j giving the beginning tangent vector for segment number j. If b = 0 and c = 0, both of these formulas become identical, and the same as (CURV3-31).

Probably the best way to understand what the constants b and c do is to experiment with them. The Excel worksheet KBSCURVE.XLS is set up to apply this model the the S-Curve example used just above to illustrate the cardinal spline formula.

However, before developing working formulas based on (CURV3-35), it's worth taking a bit closer look at the formulas themselves. First, note that when c = 0, the two formulas are identical, indicating it is the value of c here which breaks the **C**^{1} or **G**^{1} continuity. With c = 0, the formulas above simplify to:

$$D_j = s[\,(1+b)(p_j - p_{j-1}) + (1-b)(p_{j+1} - p_j)\,]$$

The first term in the square brackets is a vector pointing from p_{j-1} to p_j and the second term is a vector pointing from p_j to p_{j+1}. By varying the value of b, we make the continuous tangent at point j either more like the direction from which the curve has just come (when $b > 0$) or more like the direction in which the curve will be going (when $b < 0$). This is sometimes referred to as giving the curve a reverse or forward *bias* . Figure 13 shows the heavier curve of Figure 12, but with c = 0 and b = 0.8, and fairly pronounced reverse bias. Notice how the curve tends to go through the interpolating points and continue on in the same direction some distance before turning and heading towards the next interpolation point. The reverse bias introduces an overshoot effect (now stop thinking about cartoon characters in hot pursuit of each other and concentrate on the mathematics!). A forward bias would appear as a sort of anticipatory effect, swinging wide as the next point is approached so that the curve passes through one point heading in the general direction of the next point

The parameter c or its effect doesn't have a special name. If we drop the reference to b in (CURV3-35), we get two different formulas:

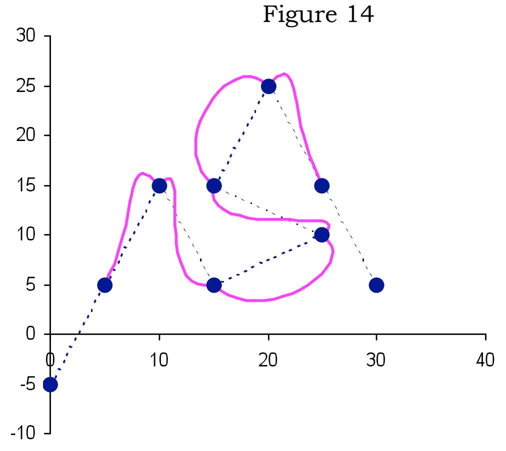
$$D_j^{out} = s[\,(1+c)(p_j - p_{j-1}) + (1-c)(p_{j+1} - p_j)\,]$$

and

$$D_j^{\,in} = s[\,(1-c)(p_j - p_{j-1}) + (1+c)(p_{j+1} - p_j)\,]$$

Relatively small negative values of c are most useful. In that case, the tangent outbound from point j has a forward bias, while the inbound tangent has a reverse bias. This will tend to give little "dimples" at the interpolating points, as illustrated in Figure 14 (which uses b = 0, but c = -0.6). You can see how the inbound curve anticipates the direction it will travel after passing through the point, whereas the outbound curve recalls the direction from which it came. In between the interpolating points, we get a smooth curve. If this curve was specifying an animation path, the interpolation points would presumably be locations where the moving object is bouncing off of walls, etc.

Figure 14



So, that's what the parameters b and c do in the model. What are the working formulas? Essentially, we just repeat the work in (CURV3-32) and (CURV3-33), but now using (CURV3-35) instead of (CURV3-31). The algebra is messy, but not impossible, and the end result is that the formula for the jth segment of the composite curve depends on the coordinates of points p_{j-1}, p_j, p_{j+1}, and p_{j+2}, so that after much simplification, we arrive at the formula

$$\mathbf{P}_j(u) = \mathbf{U}\,\mathbf{M}_{\mathbf{KB}}\,\mathbf{P_j} \qquad \text{(CURV3-36a)}$$

where **U** and $\mathbf{P}_j$ have exactly the same meaning as in formula (CURV3-33b), and the "basis" matrix $\mathbf{M}_{KB}$ is given by:

$$\mathbf{M}_{KB} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -s(1+b)(1-c) & 2s(b-c) & s(1-b)(1+c) & 0 \\ 2s(1+b)(1-c) & -3+s(1-3b+5c+bc) & 3+s(-2-4c+2bc) & -s(1-b)(1-c) \\ -s(1+b)(1-c) & 2+s(-1+b-3c-bc) & -2+s(1+b+3c-bc) & s(1-b)(1-c) \end{bmatrix} \qquad \text{(CURV3-36b)}$$

It looks pretty bad, but once you assign values to s, b, and c, each of these matrix elements reduces to an ordinary numerical value, just the same as with the simpler models.

Composite Cubic Bezier Curves

A problem with the interpolating spline formulas presented so far is that none of them have the useful property of the curve being confined within some region of space, such as the convex hull of the interpolating points. It turns out that no strictly interpolating method can have such a property, but if we relax the strict requirement of interpolation a bit, we can achieve that level of control as well.

We start here by recalling that four control points, p_0, p_1, p_2, and p_3, will give a cubic Bezier curve (see Figure 15 and the document in this series on Bezier curves for background theory). The curve will pass through the endpoints, p_0 and p_3. The direction of the curve at p_0 is along the line joining p_0 to p_1, and the direction of the curve at p_3 is along the line joining p_2 to p_3. This curve is confined within the convex hull formed by the four points.

Figure 15

You can see the connection with the Hermite approach. For this cubic segment, p_0 and p_3 give the interpolation points required by Hermite, and the directions $p_0 \rightarrow p_1$ and $p_2 \rightarrow p_3$ give the direction of the tangent vectors at p_0 and p_3. So, if instead of requiring the curve to interpolate every single point, we require it to interpolate every third point in the long run, then we have a way of specifying the geometric information required by a composite cubic Hermite curve approach, interpolation of selected points, and a sort of local convex hull property.

Just for reference's sake, the formula for the cubic Bezier segment sketched in Figure 15 is

$$\mathbf{p}(u) = \mathbf{U}\,\mathbf{M}_{\mathbf{CB}}\,\mathbf{P} = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \qquad \text{(CURV3-37)}$$

The bolding of the p's indicates, of course, that they represent rows of x, y, and possibly z coordinates in each case.

If a second segment was to be attached to the segment sketched in Figure 15, the only point that is constrained if at least $\mathbf{G}^1$ continuity is to be maintained in the next one in the sequence, p_4. For at least $\mathbf{G}^1$ continuity, p_2, p_3, and p_4 must be colinear. In general, as segments are added on to the end of a composite cubic Bezier curve, it will always be the points numbered as a multiple of 4 that are constrained in this way (p_4, p_8, p_{12}, … p_{4k}, …). If $\mathbf{G}^1$ continuity is to be enforced, it's probably best to have these points generated automatically by the program once the preceding points have been specified. We've seen the required formula before in (CURV3-34). Adapted to this situation, it becomes

$$P_{4k} = p_{4k-1} + (p_{4k-1} - p_{4k-2}) = 2p_{4k-1} - p_{4k-2} \qquad \text{(CURV3-38)}$$

Other Methods

This is where we stop our discussion of interpolating cubic splines as far as COMP 4560 is concerned – in fact, much of this material should probably be regarded as reference material because class time is unlikely to allow a thorough discussion of more than the basic ideas. There are many more methods available, variations on the methods described above, and issues that could be discussed. The intention here was to cover the basic ideas that you need to understand the broader technical literature, and to give a few effective approaches that will work adequately in many applications.

One class of methods is based on calculating a second set of control points which can be used to construct a cubic B-spline curve through the original interpolating points (see S. F. Hill, *Computer Graphics*, MacMillan, 1990; Section 14.6). Robert Kauffmann has presented a method using blending functions that involve trigonometric functions. (Dr. Dobb's Journal, May, 1997). Both of these alternatives address certain shortcomings in the interpolating cubic spline methods discussed in this document. However, the circumstances under which they are clearly advantageous are not obvious. For example, one advantage of the Kauffmann's trigonometric splines is that they are "infinitely" continuous at the joints – however, experiments with actual shapes don't usually result visibly in an even "smoother" curve than the natural cubic spline. Also, Kauffmann's trigonometric splines can be used to reproduce standard curves such as circular and elliptical arcs – something that the simple cubic polynomial (CURV3-1) cannot do exactly. However, since you need to evaluate trigonometric functions to use Kauffmann's spline formula, you would probably be just as well off to use the basic parametric equations of circles and ellipses for this purpose. Of course, using Kauffmann's trigonometric splines, you could reshape circular or elliptical arcs smoothly into other shapes.

The point here is that while there are many other methods available, they tend to be clearly superior to the basic methods described in this document only in relatively specialized situations.

Appendix I: Solving Tridiagonal Band Systems of Linear Equations

Tridiagonal band systems of equations of the sort exemplified by (CURV3-13) arise frequently in calculations involving cubic splines, curve fitting, and related problems. Fortunately, a simple set of recursive formulas can be written down for their solution. The formulas given in (CURV3-14-16) and (CURV3-18-20) are specific applications of the results developed briefly below.

These systems of equations start out with the following form:

$$\begin{array}{llllllll}
b_1x_1 & + c_1x_2 & & & & & = & r_1 \\
a_2x_1 & + b_2x_2 & + c_2x_3 & & & & = & r_2 \\
 & a_3x_2 & + b_3x_3 & + c_3x_4 & & & = & r_3 \\
 & & a_4x_3 & + b_4x_4 & + c_4x_5 & & = & r_4 \\
 & & & \ldots & & & & \ldots \\
 & & & a_jx_{j-1} + b,x_j + c_jx_{j+1} & & & = & r_j \\
 & & & & \ldots & & & \ldots \\
 & & & & a_{n-1}x_{n-1} & + \; b_{n-1}x_{n-1} & + \; c_{n-1}x_n = & r_{n-1} \\
 & & & & & a_nx_{n-1} & + \; b_nx_n = & r_n
\end{array}$$

(CURV3-39)

We will assume here that none of the coefficients a_j, b_j, or c_j are zero. When that is true, we can gain a bit of simplification by dividing the second through n^{th} equations by their a_j, so that the first term in every equation has a coefficient of 1, leading to the basic system:

$$\begin{array}{llllllll}
b_1x_1 & + c_1x_2 & & & & & = & r_1 \\
x_1 & + b_2x_2 & + c_2x_3 & & & & = & r_2 \\
 & x_2 & + b_3x_3 & + c_3x_4 & & & = & r_3 \\
 & & x_3 & + b_4x_4 & + c_4x_5 & & = & r_4 \\
 & & & \ldots & & & & \ldots \\
 & & & x_{j-1} + b,x_j + c_jx_{j+1} & & & = & r_j \\
 & & & & \ldots & & & \ldots \\
 & & & & x_{n-1} & + \; b_{n-1}x_{n-1} & + \; c_{n-1}x_n = & r_{n-1} \\
 & & & & & x_{n-1} & + \; b_nx_n = & r_n
\end{array}$$

(CURV3-40)

Here, we haven't bothered to change the form of the other symbols, considering this now to the the generic system which we will solve.

The method then proceeds as follows. First, divide the first equation by b_1, to get it into the form

$$x_1 + \gamma_1 x_2 = \delta_1 \qquad \text{(i)}$$

where, in terms of the symbols in (CURV3-40), we can write:

$$\gamma_1 = c_1/b_1 \qquad \text{and} \qquad \delta_1 = r_1/b_1 \qquad \text{(ii)}$$

Now x_1 can be eliminated from the second equation by subtracting from it this new first equation (i):

$$\begin{array}{rll} & x_1 + b_2 x_2 + c_2 x_3 & = r_2 \\ - & x_1 + \gamma_1 x_2 & = \delta_1 \end{array}$$

to give

$$(b_2 - \gamma_1)x_2 + c_2 x_3 = r_2 - \delta_1 \qquad \text{(iii)}$$

Divide this equation by $(b_2 - \gamma_1)$, to get it into the form

$$x_2 + \gamma_2 x_3 = \delta_2 \qquad \text{(iv)}$$

where

$$\gamma_2 = c_2/(b_2 - \gamma_1) \qquad \text{and} \qquad \delta_2 = (r_2 - \delta_1)/\ (b_2 - \gamma_1) \qquad \text{(v)}$$

You will see the pattern if we do one more cycle. Subtract the new second equation, (iv), from the third equation to eliminate x_2 from the third equation:

$$\begin{array}{rll} & x_2 + b_3 x_3 + c_3 x_4 & = r_3 \\ - & x_2 + \gamma_2 x_3 & = \delta_2 \end{array}$$

to give

$$(b_3 - \gamma_2)x_3 + c_3 x_4 = r_3 - \delta_2 \qquad \text{(vi)}$$

which, upon division by $(b_3 - \gamma_2)$ becomes

$$x_3 + \gamma_3 x_4 = \delta_3 \qquad \text{(vii)}$$

with

$$\gamma_3 = c_3/(b_3 - \gamma_2) \qquad \text{and} \qquad \delta_3 = (r_3 - \delta_2)/\ (b_3 - \gamma_2) \qquad \text{(viii)}$$

Thus, we will calculate the sequence of values:

$$\begin{array}{llll} \gamma_1 = c_1/b_1 & \text{and} & \delta_1 = r_1/b_1 & \text{(ii)} \\ \gamma_2 = c_2/(b_2 - \gamma_1) & \text{and} & \delta_2 = (r_2 - \delta_1)/\ (b_2 - \gamma_1) & \text{(v)} \\ \gamma_3 = c_3/(b_3 - \gamma_2) & \text{and} & \delta_3 = (r_3 - \delta_2)/\ (b_3 - \gamma_2) & \text{(viii)} \end{array}$$

and in general

$$\gamma_j = c_j/(b_j - \gamma_{j-1}) \qquad \text{and} \qquad \delta_j = (r_j - \delta_{j-1})/\ (b_j - \gamma_{j-1}) \qquad \text{(ix)}$$

for $j = 2, 3, 4, \ldots.,$ until $j = n - 1$. At this stage, the $(n-1)^{th}$ and n^{th} equations will be:

$$x_{n-1} + \gamma_{n-1} x_n = \delta_{n-1}$$

and

$$x_{n-1} + b_n x_n = r_n$$

Subtracting the former from the latter gives an equation containing just x_n, namely,

$$(b_n - \gamma_{n-1})\, x_n = r_n - \delta_{n-1} \qquad \text{(x)}$$

so

$$x_n = (r_n - \delta_{n-1})/\ (b_n - \gamma_{n-1}) \qquad \text{(xi)}$$

But, since equation (n-1) is

$$x_{n-1} + \gamma_{n-1}\, x_n = \delta_{n-1}$$

and we know x_n, we can write

$$x_{n-1} = \delta_{n-1} - \gamma_{n-1}\, x_n \qquad \text{(xii)}$$

giving the value of x_{n-1}. In fact, every remaining equation has the form

$$x_j + \gamma_j\, x_{j+1} = \delta_j$$

and so, we can work our way backwards from j = n - 2 to 1 using

$$x_j = \delta_j - \gamma_j\, x_{j+1} \qquad \text{(xiii)}$$

There's been a lot of repetition in the last page or so, to try to make the patterns and their general expression more obvious. We can now state the overall set of calculations in summary. Calculation of the solution of this system is a two-stage process. In the first stage, compute

$$\gamma_1 = c_1/b_1 \quad \text{and} \quad \delta_1 = r_1/b_1 \qquad \text{(CURV3-41a)}$$

and then, for j = 2, ..., n-1, compute

$$\gamma_j = c_j/(b_j - \gamma_{j-1}) \quad \text{and} \quad \delta_j = (r_j - \delta_{j-1})/\ (b_j - \gamma_{j-1}) \qquad \text{(CURV3-41b)}$$

Then, start the second stage by computing

$$x_n = (r_n - \delta_{n-1})/\ (b_n - \gamma_{n-1}) \qquad \text{(CURV3-41c)}$$

and then for j = n-1, ..., 1, compute

$$x_j = \delta_j - \gamma_j\, x_{j+1} \qquad \text{(CURV3-41d)}$$

These four formulas are in order formula (ii), (ix), (xi) and (xiii) above.

In the special case that all the c_j's are equal to 1 in (CURV3-40), these formulas become even simpler:

$$\gamma_1 = 1/b_1 \quad \text{and} \quad \delta_1 = r_1/b_1 = r_1\, \gamma_1 \qquad \text{(CURV3-42a)}$$

Then: $\gamma_j = 1/(b_j - \gamma_{j-1})$ and $\delta_j = (r_j - \delta_{j-1})\, \gamma_j$, $j = 2, \dots n - 1$ (CURV3-42b)

Then\` $x_n = r_n - \delta_{n-1}\, \gamma_{n-1}$ (CURV3-42c)

and $x_j = \delta_j - \gamma_j\, x_{j+1}$, $j = n – 1, \dots, 1.$ (CURV3-42d)

This set of equations (CURV3-42a, b, c, d) are the ones used to generate formulas (CURV3-14, 15, 16) and (CURV3-18, 19, 20).

Appendix II: Evaluating Cubic Polynomials

All of the cubic spline methods described so far eventually require the repeated evaluation of a cubic polynomial,

$$p(u) = au^3 + bu^2 + cu + d \qquad \text{(CURV3-1)}$$

If you code this as

A*U*U*U + B*U*U + C*U + D (CURV3-43)

then the evaluation of each coordinate of each point plotted along the spline requires six multiplications and three additions. On some systems, using the exponentiation operator,

A*U^3 + B*U^2 +C*U + D (CURV3-44)

might result in even less efficient computation if the exponentiation operation requires evaluation of a logarithm. Recall, that these formulas are repeated twice for each point in two dimensions and three times for each point in three dimensions.

You can improve on (CURV3-43) significantly by rewriting (CURV3-1) in a form commonly called **Horner's Method**, obtained by partial factoring and grouping of terms:

$$p(u) = ((au + b)u + c)u + d \qquad \text{(CURV3-45)}$$

This form gives you the value of p(u) in just three multiplications and three additions, reducing the required number of multiplications by 3 from 6. Since the multiplications are usually considered to be far more expensive than additions, method (CURV3-45) is about half as costly as method (CURV3-43).

One can do even better than this however, using a "difference" method. In this case, the so-called **forward difference method** is based on the following observations. Let δ be some step length (it can be any finite value in principle – this method gives exact results, to within normal round-off error that occurs whenever you are doing floating point calculations with a finite machine precision). Now, substituting $u + \delta$ into (CURV3-1) gives us the value of p at this parameter value as

$$p(u + \delta) = a(u + \delta)^3 + b(u + \delta)^2 + c(u + \delta) + d$$

Thus, the change in the value of p when we move from parameter value u to parameter value $u + \delta$ can be worked out to be

$$\begin{aligned}\Delta p &= p(u + \delta) - p(u)\\ &= a(u + \delta)^3 + b(u + \delta)^2 + c(u + \delta) + d - [au^3 + bu^2 + cu + d]\\ &= a(u^3 + 3u^2\delta + 3u\delta^2 + \delta^3) + b(u^2 + 2u\delta + \delta^2) + c(u + \delta) + d\\ &\qquad - au^3 - bu^2 - cu - d\\ &= u^2(3a\delta) + u(3a\delta^2 + 2b\delta) + a\delta^3 + b\delta^2 + c\delta\end{aligned}$$

Now, if we had a way of calculating Δp, then we could calculate $p(u + \delta)$ from p(u) with just a single addition(!):

$$p(u + \delta) = p(u) + \Delta p \qquad \text{(CURV3-46a)}$$

The catch is that Δp is a rather complicated expression. However, it is still a polynomial, and in fact, just a second order polynomial. So we repeat the same approach, but on Δp: The change, $\Delta^2 p$ in Δp between the parameter values u and $u + \delta$ is given by

$$\begin{aligned}\Delta^2 p &= \Delta p(\text{with parameter value } u + \delta) - \Delta p(\text{with parameter value } u)\\ &= (u + \delta)^2(3a\delta) + (u + \delta)(3a\delta^2 + 2b\delta) + a\delta^3 + b\delta^2 + c\delta\\ &\qquad - [u^2(3a\delta) + u(3a\delta^2 + 2b\delta) + a\delta^3 + b\delta^2 + c\delta]\\ &= 6a\delta^2 u + 6a\delta^3 + 2b\delta^2\end{aligned}$$

after a bit of tedious algebraic simplification. Thus, if we had a way of calculating $\Delta^2 p$, then we could get the required value of Δp with just a single addition(!):

$$\Delta p(\text{with parameter value } u + \delta) = \Delta p(\text{with parameter value } u) + \Delta^2 p \qquad \text{(CURV3-46b)}$$

Well, there's still a catch, since the formula for $\Delta^2 p$ still involves some calculation. However, the formula for $\Delta^2 p$ is in fact a first order polynomial (just the first power of u is present), which is one simpler than the formula for Δp, so progress is still being made. We apply this procedure one more time, deriving a formula for $\Delta^3 p$, the amount by which $\Delta^2 p$ changes between parameter value u and u + δ:

$$\Delta^3 p = \Delta^2 p(\text{with parameter value } u + \delta) - \Delta^2 p(\text{with parameter value } u)$$

$$= \delta^2 (u + \delta) + 6a\delta^3 + 2b\delta^2 - [\delta^2 u + 6a\delta^3 + 2b\delta^2] = 6a\delta^3,$$

which is now a constant, independent of the value of u. Thus, if we calculate the value

$$\Delta^3 p = 6a\delta^3 \qquad \text{(CURV3-46c)}$$

once, then whenever we know the value of $\Delta^2 p$ at some value of u, we can get the value of $\Delta^2 p$ at the parameter value u + δ using just a single addition:

$$\Delta^2 p(\text{with parameter value } u + \delta) = \Delta^2 p(\text{with parameter value } u) + \Delta^3 p \qquad \text{(CURV3-46d)}$$

Now, if you've been keeping track of things, you'll see that the formulas (CURV3-46a, b, c, d) give us a method of calculating p for the parameter value u + δ from the value of p for the parameter value u by doing just three additions – no multiplication at all is required! In fact, this same approach can be used to derive a method for calculating successive values of an nth degree polynomial in general which requires just n additions and no multiplication, a potentially great reduction in computational work.

To summarize the algorithm as you'd apply it in the case of a cubic polynomial, where each segment of the curve being rendered begins with u = 0, you need to first carry out an initialization step: calculate

$$\begin{aligned} p(0) &= d \qquad \text{(no calculation is necessary)} \\ \Delta p_0 &= a\delta^3 + b\delta^2 + c\delta = \delta(c + \delta(b + a\delta)) \\ \Delta^2 p_0 &= 6a\delta^3 + 2b\delta^2 = 2\delta^2 (b + 3a\delta) \\ \Delta^3 p_0 &= 6a\delta^3 \end{aligned} \qquad \text{(CURV3-47a)}$$

Depending on how you organize the calculations, this initialization step requires up to 12 multiplications. Now, the following recursion cycle will generate values of p at steps of δ in u:

$$\begin{aligned} p &\leftarrow p + \Delta p \\ \Delta p &\leftarrow \Delta p + \Delta^2 p \\ \Delta^2 p &\leftarrow \Delta^2 p + \Delta^3 p_0 \end{aligned} \qquad \text{(CURV3-47b)}$$

where $\Delta^3 p_0$ is a constant and so never needs updating once initialized for a particular polynomial and value of δ. Do the three steps of (CURV3-47b) in order, requiring three additions per cycle, and the values of p generated will be values of p for the parameter increasing in increments of δ as far as necessary. This is an exact method – there is no approximation or error present in this method other than the sort of round-off error that always occurs when doing floating point arithmetic using finite precision.

Example:
To illustrate the use of formulas (CURV3-47), we give a few steps of a sample calculation here. Suppose the cubic polynomial to be evaluated is

$$p(u) = 3u^3 + 5u^2 - 2u + 7 \qquad \text{(CURV3-48)}$$

and we wish to evaluate p for u running from 0 to 1 in ten equal steps. This means $\delta = 0.1$. Then, the initialization step gives:

$$p(0) = 7$$
$$\Delta p_0 = a\delta^3 + b\delta^2 + c\delta = 3(0.1)^3 + 5(0.1)^2 - 2(0.1) = -0.147$$
$$\Delta^2 p_0 = 6a\delta^3 + 2b\delta^2 = (6)(3)(0.1)^3 + (2)(5)(0.1)^2 = 0.118$$
$$\Delta^3 p_0 = 6a\delta^3 = (6)(3)(0.1)^3 = 0.018$$

Now, the first cycle of (CURV3-47b) gives

$$p_{new} = p_{old} + \Delta p = 7 + (-0.147) = 6.853$$
$$\Delta p_{new} = \Delta p_{old} + \Delta^2 p = -0.147 + 0.118 = -0.029$$
$$\Delta^2 p_{new} = \Delta^2 p_{old} + \Delta^3 p = 0.118 + 0.018 = 0.136$$

Notice that if you calculated p(u) directly from (CURV3-48) using u = 0.1, you would have gotten $p(0.1) = 3(0.1)^3 + 5(0.1)^2 - (2)(0.1) + 7 = 6.853$, exactly what this first cycle gave for the value of p, but with just addition, and no multiplication.

One more cycle of (CURV3-47b) in this case gives

$$p_{new} = p_{old} + \Delta p = 6.853 + (-0.029) = 6.824$$
$$\Delta p_{new} = \Delta p_{old} + \Delta^2 p = -0.029 + (0.136) = 0.107$$
$$\Delta^2 p_{new} = \Delta^2 p_{old} + \Delta^3 p = 0.136 + 0.018 = 0.154$$

and again, the first line here is confirmed by $p(0.2) = 3(0.2)^3 + 5(0.2)^2 - (2)(0.2) + 7 = 6.824$. We can continue this cycle, getting successively the values of p when u = 0.3, 0.4, 0.5, and so on as far as we wish. Thus, after an initialization step which requires some multiplications, this method gives us successive values of the cubic polynomial at the cost of just three additions per evaluation.

◆◆◆

What we've described here is just the simplest approach to using forward differences to evaluate a cubic polynomial at a uniformly-spaced sequence of values of the variable or parameter.

For reference, the forward difference method for second order polynomials

$$p(u) = au^2 + bu + c$$

is set up as

$$p(0) = c$$
$$\Delta p_0 = a\delta^2 + b\delta$$
$$\Delta^2 p_0 = 2a\delta^2$$

and so the cycling step is simply

$$p_{new} = p_{old} + \Delta p$$
$$\Delta p_{new} = \Delta p_{old} + \Delta^2 p = \Delta p_{old} + 2a\delta^2$$

giving successive values of the quadratic polynomial carrying out just two additions.

While the forward difference methods represent a considerable increase in computational efficiency as the number of times a particular cubic polynomial is to be evaluated, there are algorithms which go beyond even this is a number of ways. First, considerable time can be saved if the need to do floating point arithmetic is eliminated. Since pixel coordinates are integers, it is worth considering working directly with the integer screen coordinates of the pixels that will image the curve. Secondly, there is some advantage in being able to vary to value of δ as we move along a curve, using larger values of δ in regions where the curvature is slight and smaller values in regions where the curve is turning more sharply – methods that incorporate this feature are said to be “adaptive.” Algorithms that address these and other issues tend to be rather complicated, but worth the effort in those applications where curve generation is a major limitation on performance. We won’t discuss any details here. A paper by Benjamin Watson and Larry F Hodges (“Algorithms for Rendering Cubic Curves”) is available on the internet at http://www.gatech.edu/gvu/reports/index.html under the 1992 section. This paper describes a pair of algorithms Watson & Hodges developed and evaluate, and gives references into the more general literature on this topic.