Cox-deBoor Equations for B-Splines
AUI Course
©Denbigh Starkey

1. Background

This is a continuation of my previous set of notes, where I described three basic cubic spline systems, Hermite, Bezier, and B-Splines. I'll assume here that you understand these concepts and that I don't need to reintroduce the notation.

In these notes I'll be concentrating on B-Splines, with an emphasis on the Cox-deBoor equations that define them.

Cox and deBoor independently came up with the recursive definition of B-Spline curves in terms of a monotonically increasing knot vector, a sequence of control points, and the order of the curve. In my previous notes I briefly described knot vectors, but didn't show how they worked in general.

2. Definitions

The *order* (or, confusingly, the *degree*) of a spline curve, d, is one greater than the degree of the polynomials that define the curve segments. So, for example, if we want a spline based on cubic parametric polynomials then the order will be 4. The order is defined more intuitively as the number of control points affecting any point on the curve.

There will be $n + 1$ *control points*, $p_0, \ldots\ p_n$. In graphics we are mainly concerned with points and splines in 2D and 3D, but the principles also apply to points in 1D or in higher dimensions than 3D. For example, when we use NURBS we'll actually be using 4D spline curves which will be projected into 3D for the final result.

A *knot vector* (often shortened to *k.v.* or just *kv*) is a monotonically increasing list of floats called *knots*. So if the knot vector is $(u_0, u_1, \ldots, u_m)$ then $u_i \leq u_{i+1}$, $0 \leq i < m$. This isn't strictly monotonic, and so consecutive knot values can be equal. As I'll show when I get to the Cox-deBoor equations, the number of knots is $d + n + 1$ (i.e., the order plus the number of control points). Most graphics systems will allow bigger knot vectors than this, but only this many will be used in the computations, so I'll always assume that the knot vector has this size.

A *uniform knot vector* has equal separation between the knots. The most common form begins at 0 and increases by 1 and so (0, 1, 2, 3, 4, 5) is the standard uniform knot vector of length 6. A normalized uniform knot vector begins at 0 and ends at 1 and so normalizing the previous knot vector gives the knot vector (0, 0.2, 0.4, 0.6, 0.8, 1.0).

An *open uniform knot vector* isn't uniform, since in standard form it begins with d zeros (where d is the order) then increases uniformly, usually with a separation of 1, until ending with d copies of the last value. The knot vector is said to have a *multiplicity* of d at each end. E.g., for a quadratic (order 3) 二次 B-spline with six control points the standard open uniform knot vector will be (0, 0, 0, 1, 2, 3, 4, 4, 4) since it must have length 3 + 6. Normalized it will be (0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1).

3. Cox-deBoor Equations

The definition of a spline curve is given by:

$$P(u) = \sum_{k=0}^{n} p_k B_{k,d}(u)$$

where d is the order of the curve and the *blending functions* $B_{k,d}(u)$ are defined by the recursive Cox-deBoor equations:

$$B_{k,1}(u) = \begin{cases} 1 & \text{if } u_k \le u \le u_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

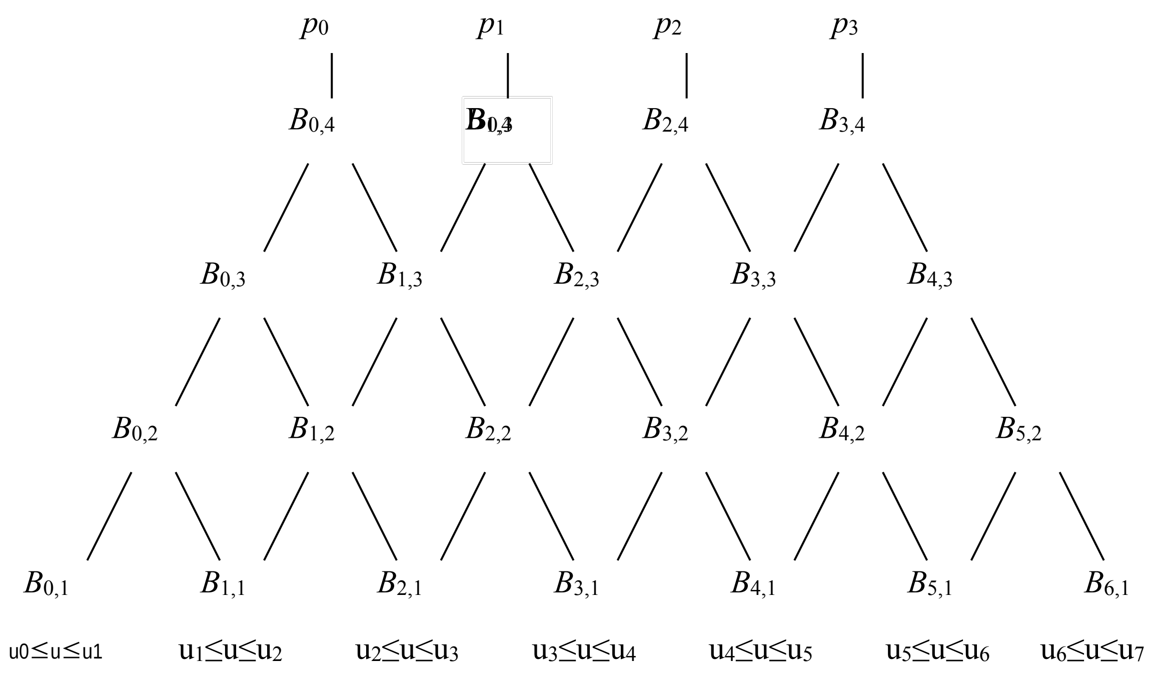
$$B_{k,\mathrm{d}}(u) = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u), \quad d > 1$$

The generated curve is defined as being the part that is in the range of d blending functions of the form $B_{k,1}(u)$ and u values outside this range are not used. This means that all points in the curve are controlled by d (the order) control points.

I find it useful to look at these equations as a tree of $B_{k,d}$'s.

As an example, in the tree on the next page I've shown a four control point cubic system (so its order is 4). It shows the four control points, with their associated summation B values at the top. At the bottom it shows the knot ranges where each $B_{k,1}$ is defined. Each $B_{k,d}$ value higher in the tree is linked to the $B_{k,d-1}$ values that are used in its definition.

I'll have additional uses for the tree later, but now it can be used to confirm some properties of B-splines. One is the number of knot values needed; here we have $d = 4$ and four control points, and looking at the knots that are used in the definition they range from u_0 to u_7, and so there are 8 knots. Since $4 + 4 = 8$ this satisfies our length rule. Obviously as additional control points are added to the system we will need an equal number of additional knots at the bottom. Another property can be seen by selecting a control point and looking at the knot ranges that it affects. E.g., tracking down through the tree shows that p_1 affects the four ranges from u_1 to u_5.



For this tree the generated curve will be in the parameter range $u_3 \le u \le u_4$, since this is the only parameter range where all four ($d = 4$) of the control points have influence.

4. Using $u_k \le u \le u_{k+1}$ or $u_k \le u < u_{k+1}$

The basis of the recursive definition of the blending functions given above was:

$$B_{k,1}(u) = \begin{cases} 1 & \text{if } u_k \le u \le u_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

You'll also see this with a < for the last comparison as in:

$$B_{k,1}(u) = \begin{cases} 1 & \text{if } u_k \le u < u_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

There are advantages and disadvantages to both approaches. E.g., later in these notes I'll have a case where, using the ≤ form I have:

$$B_{1,2}(u) = \begin{cases} 0 & \text{if } u < 1 \text{ or } u > 3 \\ u-1 & \text{if } 1 \le u \le 2 \\ 3-u & \text{if } 2 \le u \le 3 \end{cases}$$

which is a bit offensive with the double specification on what to do when $u = 2$, but gets away with it because both specifications are equal at that point. If I'd been using the < form, I'd have had a different problem because it would have led to:

$$B_{1,2}(u) = \begin{cases} 0 & \text{if } u < 1 \text{ or } u \ge 3 \\ u-1 & \text{if } 1 \le u < 2 \\ 3-u & \text{if } 2 \le u < 3 \end{cases}$$

Which although it works out well here, I really want the $3 - u$ piece to be true for $2 \le u \le 3$. So both versions have problems here that need to be patched, which in the ≤ case involve the limit as $u \to u_{d+n}$ types of fixups.

The major advantage of the < form comes when we have a knot vector that includes two or more equal knots. With the ≤ form this leads to divide by zero situations like:

$$B_{1,2}(u) = \begin{cases} 0 & \text{if } u < 1 \text{ or } u > 1 \\ \dfrac{u-1}{0} & \text{if } 1 \le u \le 1 \end{cases}$$

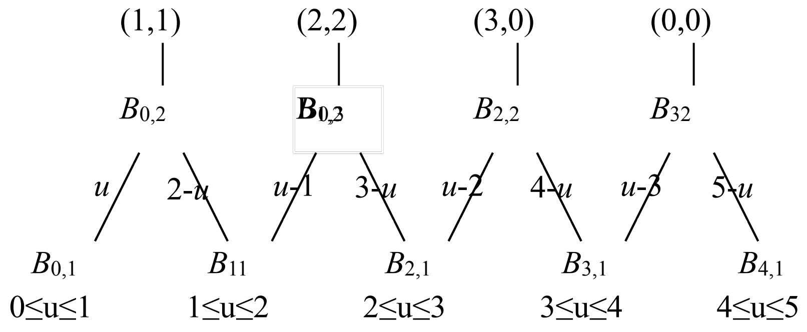
which is obviously very ugly, and is avoided in the < form, where it will become:

$$B_{1,2}(u) = \begin{cases} 0 & \text{if } u < 1 \text{ or } u \ge 1 \\ \dfrac{u-1}{0} & \text{if } 1 \le u < 1 \end{cases}$$

which is weird, but avoids the problem. To fix this up in the ≤ form we need to define that $\frac{0}{0} = 0$, which solves the problem but is obviously a kludge. To get you comfortable with both versions I'll use the ≤ form in these notes, where I'll assume that the $\frac{0}{0} = 0$ equality is true, and will use the < form in my NURBS notes.

5. Example 1: Linear Spline, Four Control Points, Uniform K.V.

I'll start with a relatively simple example where $d = 2$, we have four control points, (1, 1), (2, 2), (3, 0), and (0, 0), and we use the standard uniform knot vector (0, 1, 2, 3, 4, 5). This should give a linear spline, and each point on the spline will depend on $d = 2$ control points. The tree is shown below:



I've now put the point values at the top of the tree, and the actual knot values at the bottom of the tree. However the biggest difference here is that I have labeled the edges with their multipliers. I'll use one pair of these edges as an example, and let you confirm the others.

Consider $B_{1,2}(u)$. The Cox-deBoor equations say that this is defined by:

$$B_{1,2}(u) = \frac{u - u_1}{u_2 - u_1} B_{1,1}(u) + \frac{u_3 - u}{u_3 - u_2} B_{2,1}(u).$$

For the standard knot vector that we are using, $u_1 = 1$, $u_2 = 2$, and $u_3 = 3$, and so this becomes:

$$B_{1,2}(u) = (u - 1)B_{1,1} + (3 - u)B_{2,1},$$

and so in the tree the edges from $B_{1,2}$ to $B_{1,1}$ and $B_{2,1}$ are labeled $u - 1$ and $3 - u$, respectively.

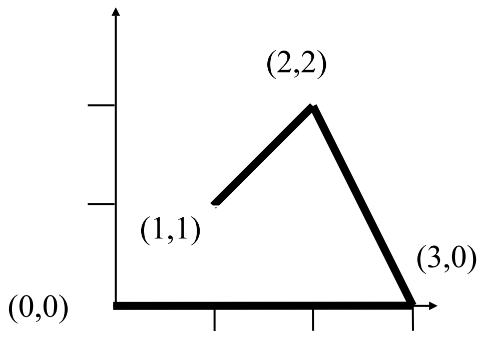
In more traditional notation,

$$B_{1,2}(u) = \begin{cases} 0 & \textit{if } u < 1 \textit{ or } u > 3 \\ u - 1 & \textit{if } 1 \le u \le 2 \\ 3 - u & \textit{if } 2 \le u \le 3 \end{cases}$$

Using the tree, we see that the range of *u* values which are under the influence of two (because $d = 2$) control points is $1 \le u \le 4$ and within this range the curve is defined by:

$$P(u) = \begin{cases} (2-u)(1,1) + (u-1)(2,2) & 1 \le u \le 2 \\ (3-u)(2,2) + (u-2)(3,0) & 2 \le u \le 3 \\ (4-u)(3,0) + (u-3)(0,0) & 3 \le u \le 4 \end{cases}$$

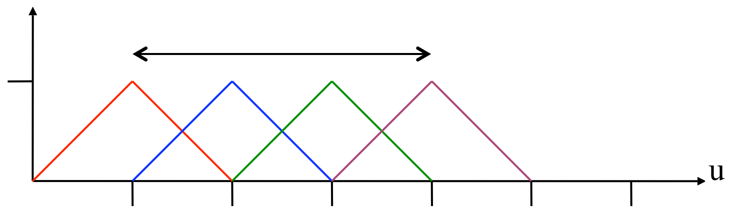
which connects the four points with straight lines, as we'd expect for a linear spline.



One more point to note before moving on to the next example. If you look at the blending functions in the definition of $P(u)$ you will see that the sum of them, for any *u*, is 1, and each blending function is ≥ 0. E.g., for the range $3 \le u \le 4$, $(4 - u) + (u - 3) = 1$ and both $(4 - u)$ and $(u - 3)$ are ≥ 0. This will always be true and gives the convex hull property for B-splines. Check it in my following four examples.

6. Blending Functions for Example 1

It can sometimes make it easier to get a handle on B-splines if we look at a graphical representation of the blending functions. I've shown this below for the previous example:



The red shape is a plot of the non-zero part of $B_{0,2}$, the blue of $B_{1,2}$ (which I computed earlier), the green of $B_{2,2}$, and the plum of $B_{3,2}$. The black double arrow shows the range of u values used in the spline.

At any u value in the relevant range, $1 \le u \le 4$, this shows that it is based on two of the control points which are associated with each $B_{i,2}$ function. It also shows that at, say, $u = 1.5$, the curve will have a value of $.5p_0 + .5p_1$.

One property that can be seen here, which will apply to any uniform spline, is that each blending function is identical to the previous function, but shifted one to the right. So once one has computed $B_{0,d}(u)$ then the remaining functions $B_{i,d}(u)$ can be computed very efficiently by substituting $(u - i)$ for u. Here $B_{0,2}$ can be computed as:

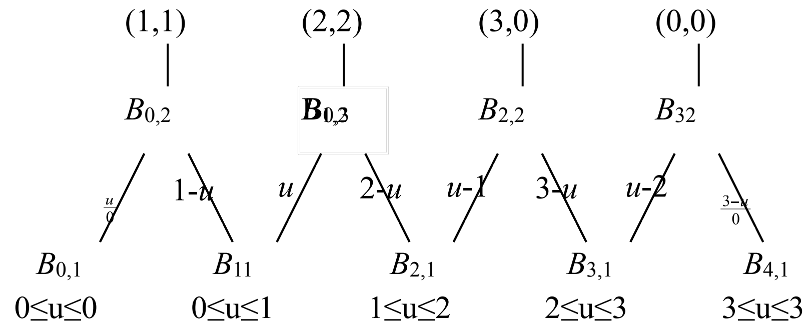
$$B_{0,2}(u) = \begin{cases} 0 & \text{if } u < 0 \text{ or } u > 2 \\ u & \text{if } 0 \le u \le 1 \\ 2 - u & \text{if } 1 \le u \le 2 \end{cases}$$

which, with the substitution $(u - 1)$ for u, gives the value that I had earlier:

$$B_{1,2}(u) = \begin{cases} 0 & \text{if } u < 1 \text{ or } u > 3 \\ u - 1 & \text{if } 1 \le u \le 2 \\ 3 - u & \text{if } 2 \le u \le 3 \end{cases} .$$

7. Example 2: Linear Spline, Four Control Points, Open Uniform K.V.

Now take the same four control points and order, but use the standard open uniform knot vector instead. In this case the vector will be (0, 0, 1, 2, 3, 3). The tree becomes:



The first thing to notice is that two of the multipliers include a divide of zero by zero. Fortunately they lead to the two *B* values that are not included in the curve definition, and so I don't really need the $\frac{0}{0} = 0$ assumption, but I'd have used it if necessary. This will always occur with open uniform knot vectors or anywhere else where there are repeated knots. From now on when labeling edges I will omit those that lead to the $(d - 1)$ *B* values on each end, where *d*, as usual, is the order.

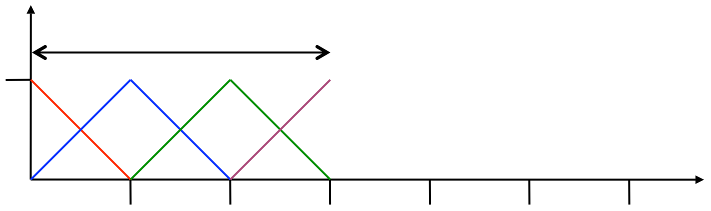
From the tree we get the following definition for $P(u)$:

$$P(u) = \begin{cases} (1-u)(1,1) + u(2,2) & 0 \le u \le 1 \\ (2-u)(2,2) + (u-1)(3,0) & 1 \le u \le 2 \\ (3-u)(3,0) + (u-2)(0,0) & 2 \le u \le 3 \end{cases}$$

which gives the same three lines connecting the points that we had before. As we'll see below, for $d > 2$ the open uniform and uniform knot vectors will usually produce very different results.

8. Blending Functions for Example 2

Graphically the blending functions for Example 2 (using the definition that $\frac{0}{0} = 0$) are shown below:



with the same color and arrow coding that I used earlier.

9. Example 3: Cubic Spline, Four Control Points, Uniform K.V.

My third example will have four control points and order 4, so will be a cubic B-spline. The tree will be the one shown in the section on the Cox-deBoor equations.

First I'll use the standard uniform knot vector, which will have length 8 because 4 + 4 = 8, and so will be (0, 1, 2, 3, 4, 5, 6, 7). Looking at the tree note that only $B_{3,1}$ uses four control points, and so the only parameter range that is relevant is $3 \le u \le 4$.

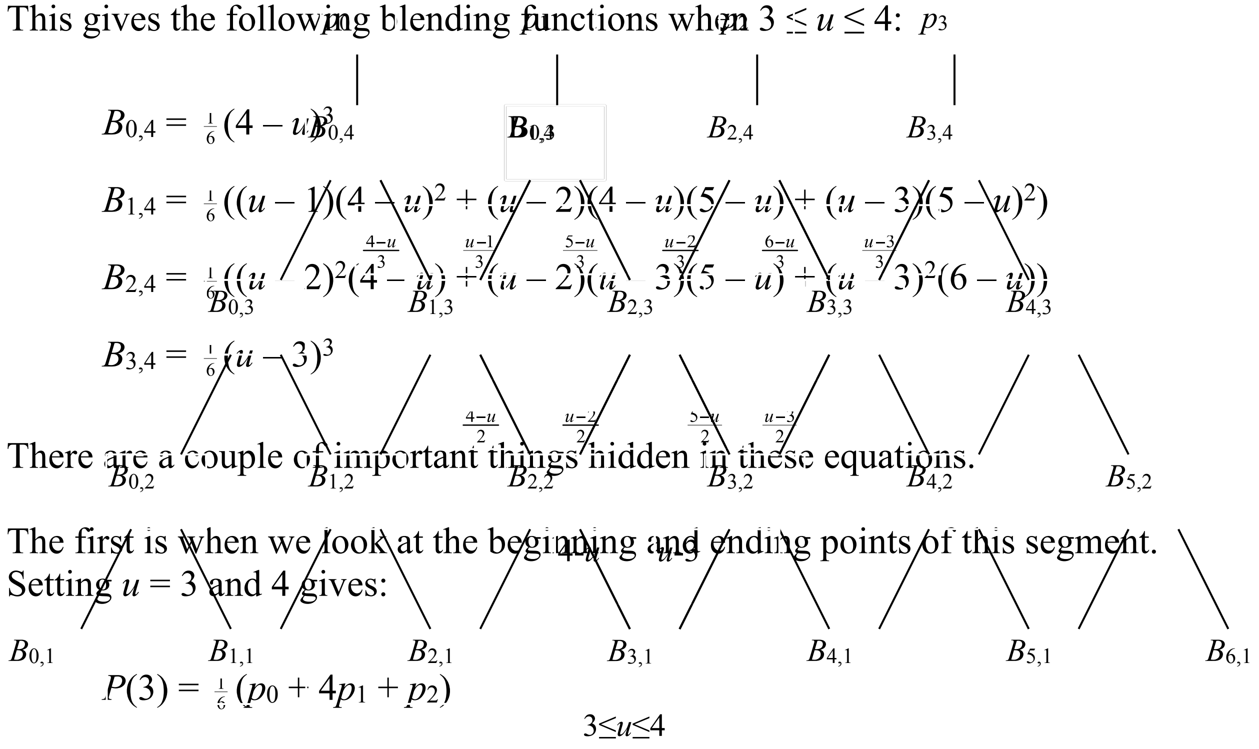
This gives the following blending functions when $3 \le u \le 4$:

$$B_{0,4} = \tfrac{1}{6}(4-u)^3$$

$$B_{1,4} = \tfrac{1}{6}((u-1)(4-u)^2 + (u-2)(4-u)(5-u) + (u-3)(5-u)^2)$$

$$B_{2,4} = \tfrac{1}{6}((u-2)^2(4-u) + (u-2)(u-3)(5-u) + (u-3)^2(6-u))$$

$$B_{3,4} = \tfrac{1}{6}(u-3)^3$$



There are a couple of important things hidden in these equations.

The first is when we look at the beginning and ending points of this segment. Setting $u = 3$ and 4 gives:

$$P(3) = \tfrac{1}{6}(p_0 + 4p_1 + p_2)$$

$$P(4) = \tfrac{1}{6}(p_1 + 4p_2 + p_3)$$

which should be very familiar after my first set of notes on splines.

Another appears if we change the parametric range from [3, 4] to [0, 1], which we can do by setting a new parameter $v = u - 3$ and so $0 \le v \le 1$. The equations will now become:

$$B_{0,4} = \tfrac{1}{6}(1-v)^3$$

$$B_{1,4} = \tfrac{1}{6}((v+2)(1-v)^2 + (v+1)(1-v)(2-v) + v(2-v)^2)$$

$$B_{2,4} = \tfrac{1}{6}((v+1)^2(1-v) + v(v+1)(2-v) + v^2(3-v))$$

$$B_{3,4} = \tfrac{1}{6}v^3$$

Multiplying these out and changing v back to u for tradition sake, we get:

$$B_{0,4} = \tfrac{1}{6}(-u^3 + 3u^2 - 3u + 1)$$

$$B_{1,4} = \tfrac{1}{6}(3u^3 - 6u^2 + 4)$$

$$B_{2,4} = \tfrac{1}{6}(-3u^3 + 3u^2 + 3u + 1)$$

$$B_{3,4} = \tfrac{1}{6}(u^3)$$

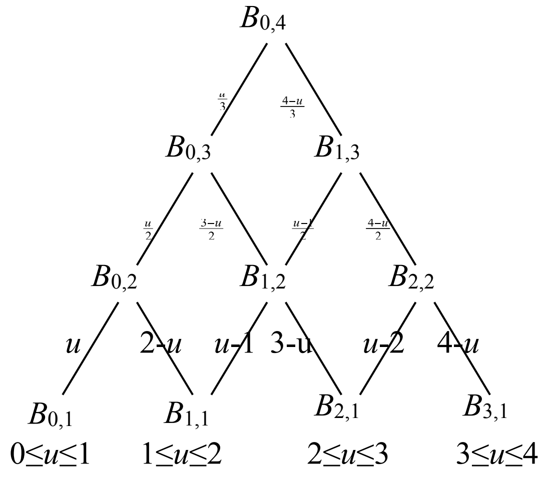
and so:

$$P(u) = [u^3\ u^2\ u\ 1]\ \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

which is the formula that I previously gave for cubic B-spline segments, and have now finally justified.

10. Blending Functions for Example 3

Since this example used a uniform knot vector, we just need to compute $B_{0,4}$ and then use shifts to get the other three top level blending functions. Looking at the subtree for $B_{0,4}$ we get:



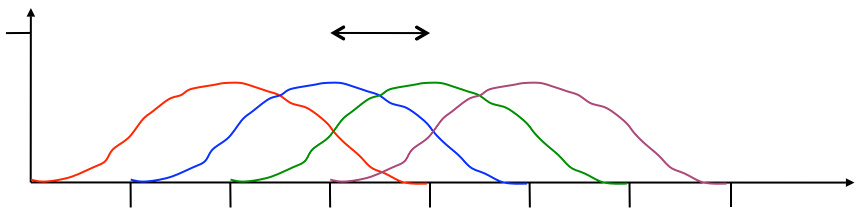
which gives:

$$B_{0,4} = \begin{cases} 0 \quad u < 0 \; or \; u > 4 \\ \frac{1}{6}u^3 \quad 0 \le u \le 1 \\ \frac{1}{6}(u^2(2-u) + u(u-1)(3-u) + (u-1)^2(4-u) \quad 1 \le u \le 2 \\ \frac{1}{6}(u(3-u)^2 + (u-1)(3-u)(4-u) + (u-2)(4-u)^2 \quad 2 \le u \le 3 \\ \frac{1}{6}(4-u)^3 \quad 3 \le u \le 4 \end{cases}$$

Now, for example, we can compute $B_{1,4}$ by substituting $(u-1)$ for u giving:

$$B_{1,4} = \begin{cases} 0 \quad u < 1 \; or \; u > 5 \\ \frac{1}{6}(u-1)^3 \quad 1 \le u \le 2 \\ \frac{1}{6}((u-1)^2(3-u) + (u-1)(u-2)(4-u) + (u-2)^2(5-u) \quad 2 \le u \le 3 \\ \frac{1}{6}((u-1)(4-u)^2 + (u-2)(4-u)(5-u) + (u-3)(5-u)^2 \quad 3 \le u \le 4 \\ \frac{1}{6}(5-u)^3 \quad 4 \le u \le 5 \end{cases}.$$

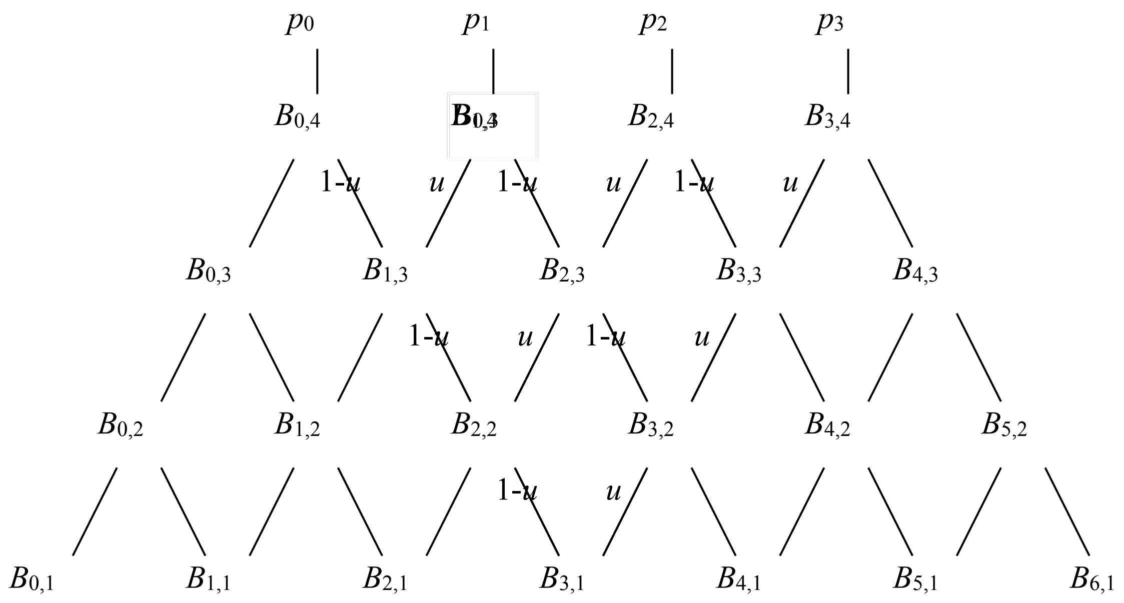
Graphically the four blending functions look like the following rough sketch:



For curves with a reasonably large number of knots (which is the usual situation) doing the computations this way can significantly improve computation times. While I have only discussed this for uniform knot vectors, it works equally well for the uniform part of splines defined using open uniform knot vectors, which means that it will work effectively at all places except for the ends. If you look back at the graphical representation of the blending functions from Example 2 you will see that the green blending function is one to the right of the blue one, and so satisfies the rule. If instead of four control points there I'd used 50, then 48 of them would have been identical apart from the x shifts, since $d = 2$.

11. Example 4: Cubic Spline, Four Control Points, Open Uniform K.V.

My fourth example will be the same as the one that I've just done except that I'll use the standard open uniform knot vector instead of the standard uniform knot vector. So we need eight elements in the vector, which will be (0, 0, 0, 0, 1, 1, 1, 1). First, as usual, I'll develop the blending function tree:



p_0 p_1 p_2 p_3
$B_{0,4}$ $B_{0,4}$ $B_{2,4}$ $B_{3,4}$
1-u u 1-u u 1-u u
$B_{0,3}$ $B_{1,3}$ $B_{2,3}$ $B_{3,3}$ $B_{4,3}$
1-u u 1-u u
$B_{0,2}$ $B_{1,2}$ $B_{2,2}$ $B_{3,2}$ $B_{4,2}$ $B_{5,2}$
1-u u
$B_{0,1}$ $B_{1,1}$ $B_{2,1}$ $B_{3,1}$ $B_{4,1}$ $B_{5,1}$ $B_{6,1}$

$0 \leq u \leq 1$

This gives relatively simple blending functions in the range $0 \leq u \leq 1$:

$$B_{0,4} = (1 - u)^3$$
$$B_{1,4} = 3u(1 - u)^2$$
$$B_{2,4} = 3u^2(1 - u)$$
$$B_{3,4} = u^3$$

From this we get:

$$P(0) = p_0$$
$$P(1) = p_3$$

and so with the open uniform knot vector the curve begins at the first control point and ends at the last control point, which will always be true, and is the main reason for using this knot vector.

导数

Differentiating gives:

$$P'(u) = -3(1-u)^2 p_0 + (3(1-u)^2 - 6u(1-u))p_1 + (6u(1-u) - 3u^2)p_2 + 3u^2 p_3$$

and so at the end points we get:

$$P'(0) = 3(p_1 - p_0)$$
$$P'(1) = 3(p_3 - p_2)$$

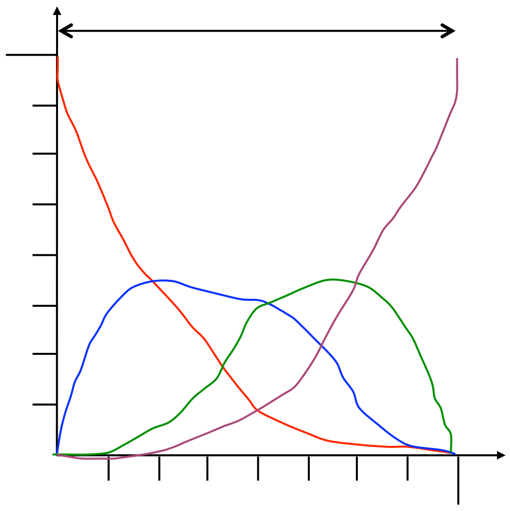
which are the same properties that are satisfied by Bezier. Putting our B-spline blending function values in matrix form we get:

$$M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

which is the Bezier form. So an open uniform cubic spline over four control points is just a cubic Bezier.

12. Blending Functions for Example 4

This is probably going to be a pretty poor diagram. The blending functions all are defined between 0 and 1, and we get something like:

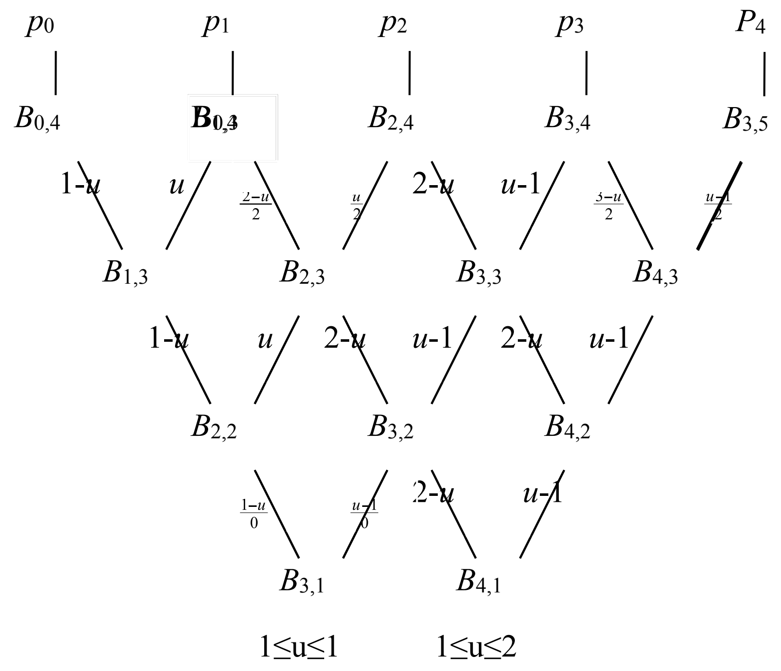


where the short grid tics are at $\frac{1}{8}$ intervals. Note that here we don't have intermediate knots and so the shift to the right rule doesn't apply.

13. Example 5: Cubic Spline, Five Control Points, Non-Uniform K.V.

I'll do a final example just to show what happens when we have a non-uniform knot vector, and also to do a cubic with more than the minimum number of control points.

In the tree below I have five control points and the order is 4 (cubic spline). Assume that the knot vector is (0, 0, 0, 1, 1, 2, 2, 3, 3). I've drawn only part of the tree, covering the areas that are influenced by four control points.



Now, for the first time, I have a couple of $\frac{0}{0}$ values in the relevant part of the tree, but I avoid the problem by just using the definition that this gives 1 as its result. The generated curve then becomes, for $1 \le u \le 2$,

$$P(u) = \frac{(2-u)^3}{2} p_1 + \left(\frac{u(2-u)^2}{2} + (u-1)(2-u)^2\right) p_2$$

$$+ \left(2(u-1)^2(2-u) + \frac{(3-u)(u-1)^2}{2}\right) p_3 + \frac{(u-1)^3}{2} p_4$$

The main lesson to learn out of this is to be careful when using a non-standard knot vector. Here p_0 is being ignored, and at the extreme u values we get:

$$P(1) = \tfrac{1}{2}(p_1 + p_2)$$

$$P(2) = \tfrac{1}{2}(p_3 + p_4)$$

which isn't very intuitive.

Usually non-standard knot vectors will be used to attract the curve closer to specific control points, but as this example shows it tends to be hard to control. A better solution to this is to use a rational spline, which I'll describe in my next set of notes, where each control point has a weight and increasing the weight associated with a control point pulls the curve closer to that point.

The blending curves for this example are a bit of a mess, so I won't show them graphically.

14. Interpolating B-Splines

There are obviously occasions when we want the generated spline to go through the control points, and there are ways to ensure that this happens. As I have shown in two examples, the open uniform knot vector makes the curve accurately start at p_0 and end at p_n, which it accomplishes by having d (the order) copies of the first and last knot value, which is called a multiplicity of d. Within the curve we can use multiple knots to get closer to control points, where a multiplicity of $(d - 1)$ will interpolate a knot under the correct circumstances. So it takes a multiplicity of d at the endpoints and $(d - 1)$ in the interior, if the knots and control points can be aligned correctly.

Say that one has control points p_0 to p_n, as usual, and one wants to create a quadratic curve that goes through all of them. First add n more control points, q_1 to q_n, where each q_i lies between $p_{i\text{-}1}$ and p_i. Now the knot vector $(0, 0, 0, 1, 1, 2, 2, \ldots, n - 1, n - 1, n, n, n)$ will generate the curve through all of the p_i's.

I'll look at an example of this in my notes on NURBS, where I use the points (1, 0), (1, 1), (0, 1), (-1, 1), (-1, 0), (-1, -1), (0, -1), (1, -1), (1, 0) and the knot vector (0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4) to get a closed quadratic spline through the five points (1, 0), (0, 1), (-1, 0), (0, -1), and (1, 0) but if you want to be convinced try working this out now.

15. Facts About B-Splines

Some of these I've shown, others I'm just claiming. I'll use *d*, as usual, for the order of the spline.

- The size of the knot vector is the sum of the order and the number of control points.
- The order cannot be greater than the number of control points.
- The most common knot vectors are the uniform and open uniform knot vectors.
- End point multiplicity of *d* at an end of a knot vector forces that end to go through the control point.
- Multiplicity within a knot vector pulls the curve closer to (or all the way to) one or more control points.
- When correctly matched, knot multiplicity of $(d - 1)$ will force the curve through an internal control point.
- There is local control since changing a control point alters only *d* curve segments.
- The continuity of the curve is $C^{(d-2)}$. So to get $C^{(1)}$ (smooth) continuity we need at least $d = 3$.
- Bezier can be implemented as a $d = 3$ open uniform B-spline without intermediate knots.
- The convex hull property holds over every consecutive *d* control points (this can be proved by induction from the Cox-deBoor equations). This because blending functions are ≥ 0 at all *u*, and for any *u* the sum of the blending functions is 1.
- In uniform parts of a knot vector consecutive blending functions will have identical shape, with a shift of one knot increment.