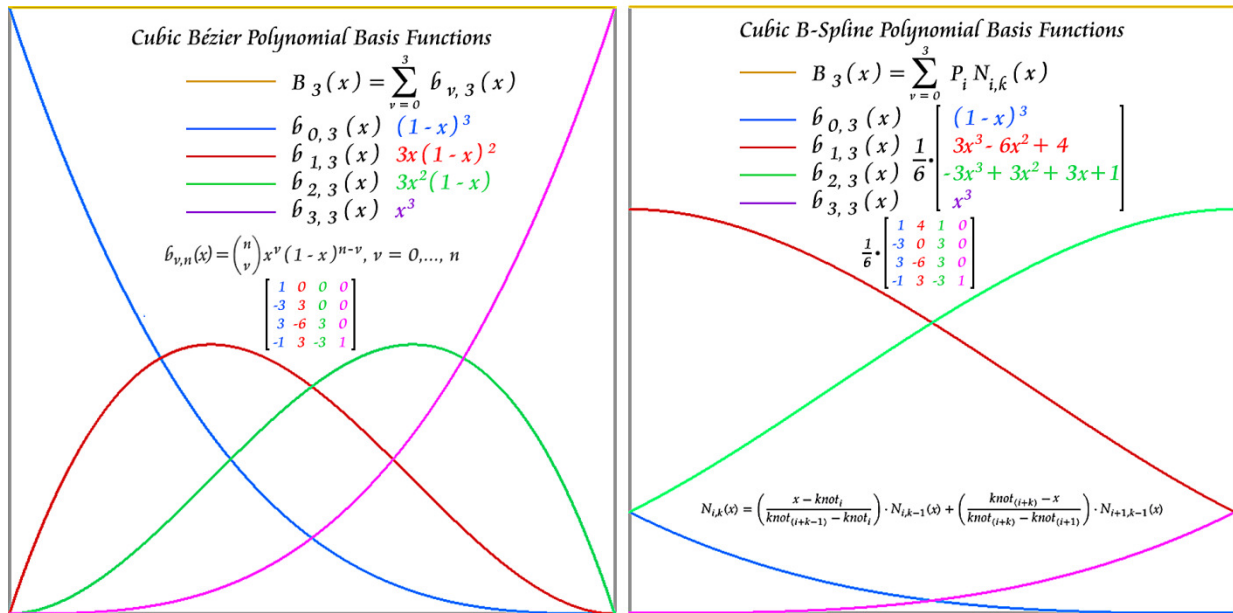


QS B-Spline and Bézier Interpolation

Michael Sargent – January 2020

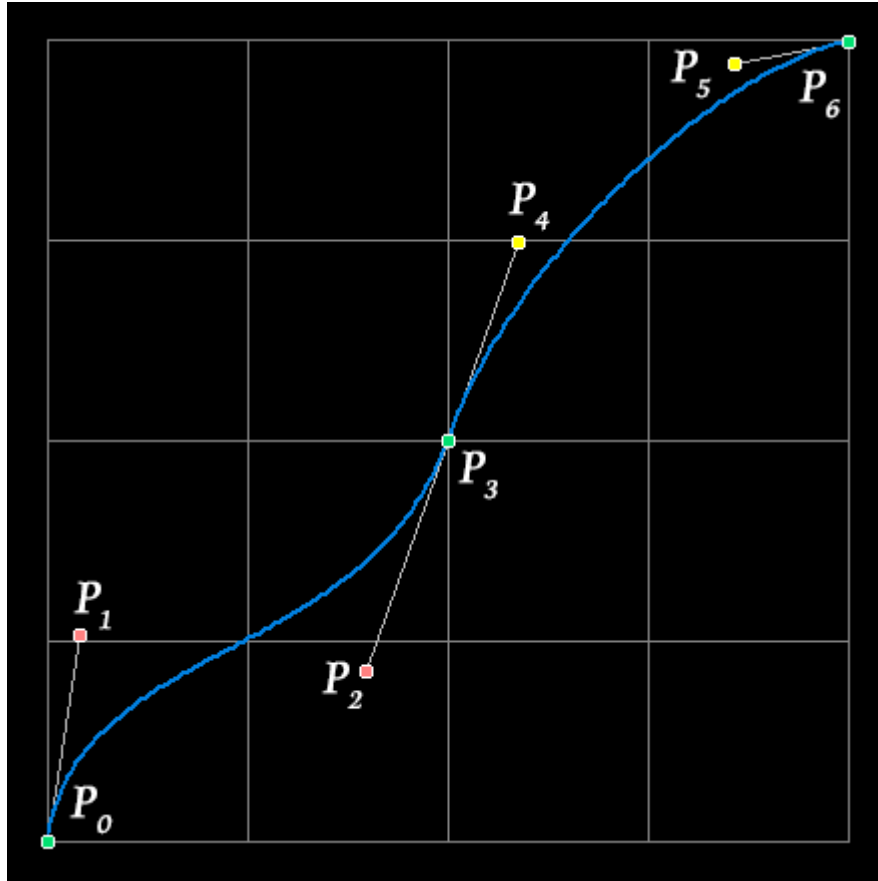
Both Bézier and B-spline curves can be characterized by basis functions. This description focuses on cubic curves, but the concepts generalize to other degrees. The basis functions of Bézier curves are a form of Bernstein equations. They can be expressed as binomial expansions, with a corresponding matrix, as shown in the diagram. B-spline curves also are characterized by basis functions, but these are derived from a more complicated recursive function $N_{i,k}$, which will be examined on page 2.



A cubic Bézier curve is generated from 4 control points, $P_0 - P_3$, which serve as weighting factors for the basis functions:

$$B(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

The curve starts at P_0 and ends at P_3 but does not pass through P_1 and P_2 unless all 4 points are colinear. Individual piecewise curves can be connected by setting P_3 of the first curve equal to P_0 of the next. Therefore, a connected curve with n components will have $3n + 1$ control points. To avoid kinks, the first derivatives of the 2 piecewise curves at their junction must be equal. This is demonstrated graphically by the familiar “handles” used in software that offers vector drawing options. Here is a Bézier curve with 2 components and 7 control points:



A cubic B-Spline curve also is generated from 4 control points:

$$B(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \cdot \frac{1}{6} \cdot \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

The basis functions are derived from the recursive function $N_{i,k}$ referred to on page 1:

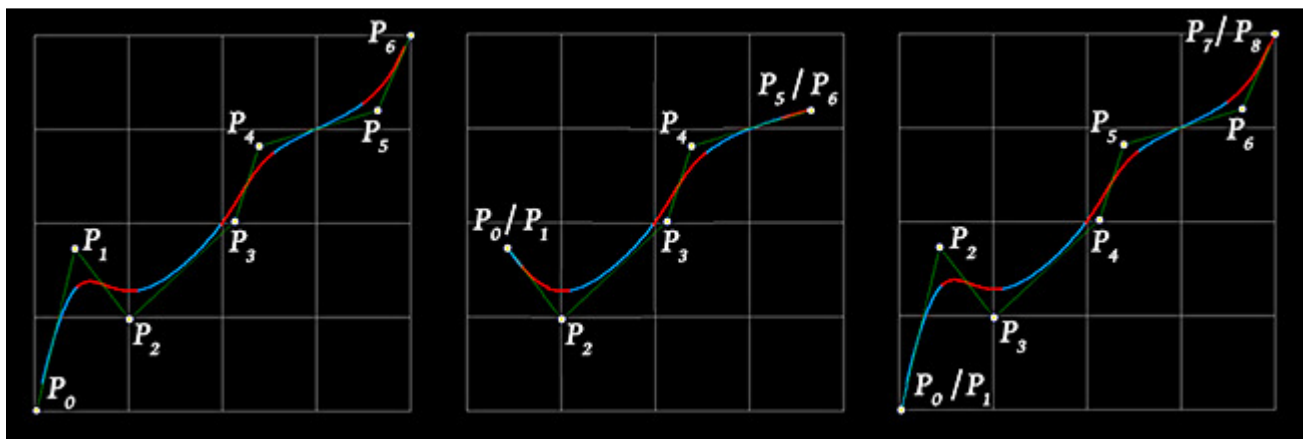
$$N_{i,k}(t) = \left(\frac{t - \text{knot}_i}{\text{knot}_{(i+k-1)} - \text{knot}_i} \right) \cdot N_{i,k-1}(t) + \left(\frac{\text{knot}_{(i+k)} - t}{\text{knot}_{(i+k)} - \text{knot}_{(i+1)}} \right) \cdot N_{i+1,k-1}(t)$$

A curve of degree d with n control points will have $(d + n + 1)$ “knots”, or transitions where a control point starts or stops influencing the curve. The variable k is an index for the $(d + n)$ knot intervals. The “knot vector” is a table of *relative* knot intervals. Typically the first and last 3 intervals are set to 0 and the rest are “uniform”, enabling a curve to be drawn that starts and ends on control points P_1 and $P_{(n-1)}$. Equivalent knot vectors for 7 control points might be:

$$[0 \ 0 \ 0 \ 0 \ 1 \ 2 \ 3 \ 4 \ 4 \ 4 \ 4] \quad \text{or} \quad [0 \ 0 \ 0 \ 0 \ 2 \ 4 \ 6 \ 8 \ 8 \ 8 \ 8]$$

An algorithm developed by Maurice Cox¹ and refined by Carl deBoor^{2,3} can be used to evaluate the $N_{i,k}$ function and derive the basis functions and matrix. (See the Appendix.)

Unlike Bézier curves, the actual B-spline curves range from the middle 2 of each set of 4 knots, so that a connected B-spline is a smooth interpolation of every possible curve generated from 4 consecutive control points. So, the curve will be drawn by (P_0, P_1, P_2, P_3) , then (P_1, P_2, P_3, P_4) , (P_2, P_3, P_4, P_5) , ... $(P_{(n-4)}, P_{(n-3)}, P_{(n-2)}, P_{(n-1)})$, whereas Bézier curves are drawn by (P_0, P_1, P_2, P_3) , then (P_3, P_4, P_5, P_6) , The B-spline on the left has 7 control points, with 6 curve segments. The curve does not start at P_0 or end at P_6 . To obtain a curve that starts and ends at control points, either P_0 can be set equal to P_1 , and P_5 to P_6 , (middle curve) or an extra control point can be added at each end and set equal to the adjacent points (right curve). This is enabled by the extra knots referred to above.



Both the Bézier and B-spline curves described so far have been generated by control points that are known (or potentially selected by software users). These control points can be inserted into the respective formulae over the range of $0 \leq t \leq 1$ in order to calculate the points that constitute the curves. If the control points are considered to be data points, the curves represent approximations of the smoothest path between those points. Although they will not actually pass through most of those points, as can be seen in the above diagrams, they allow reasonable interpolation of other data points in between the designated points.

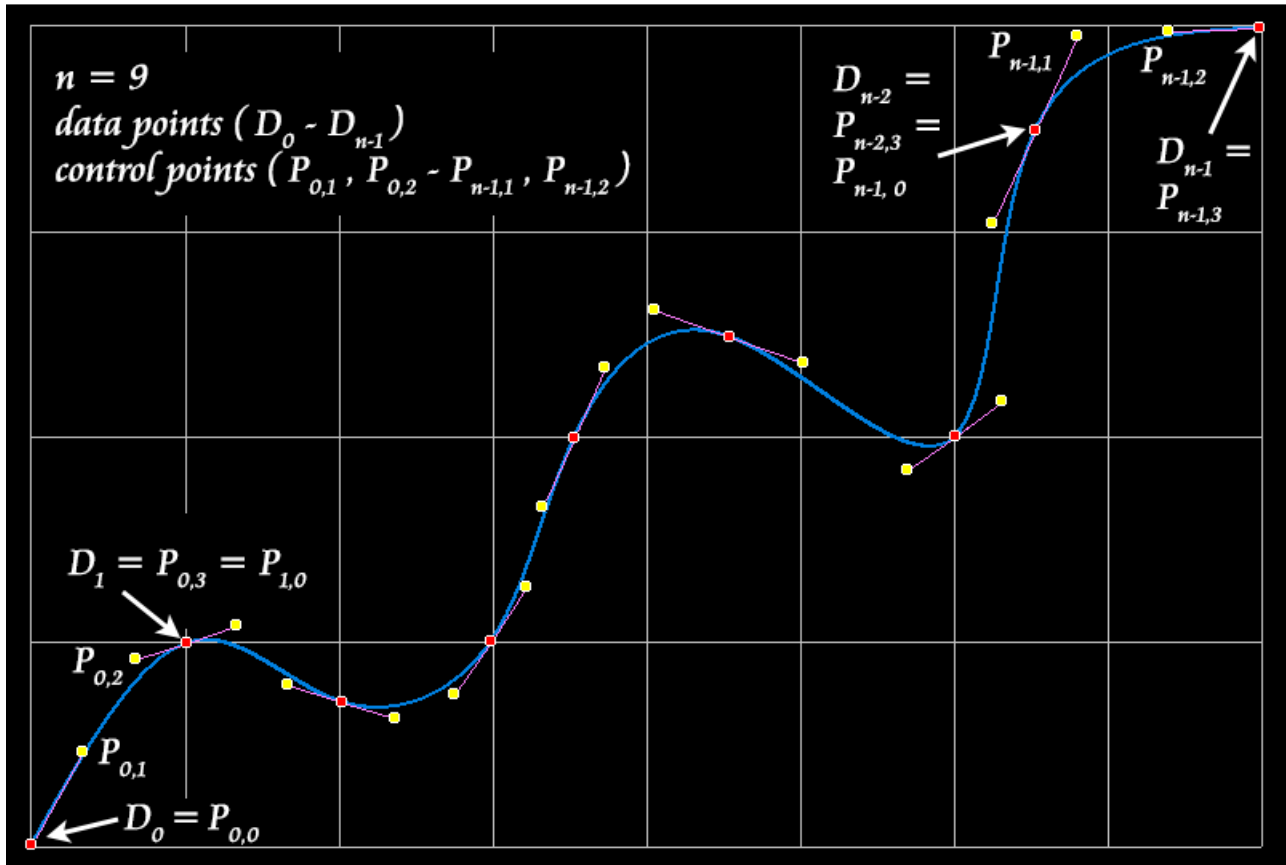
However, sometimes the opposite procedure is desired: construction of a curve that connects every member of a set of data points and that consists of interpolated points which lie along the course of the curve. This procedure is somewhat more complex and requires calculating the control points that will generate the curve that connects the data points.

¹ Cox, Maurice G. *The numerical evaluation of B-splines.*, IMA Journal of Applied Mathematics 10.2 (1972): 134-149.

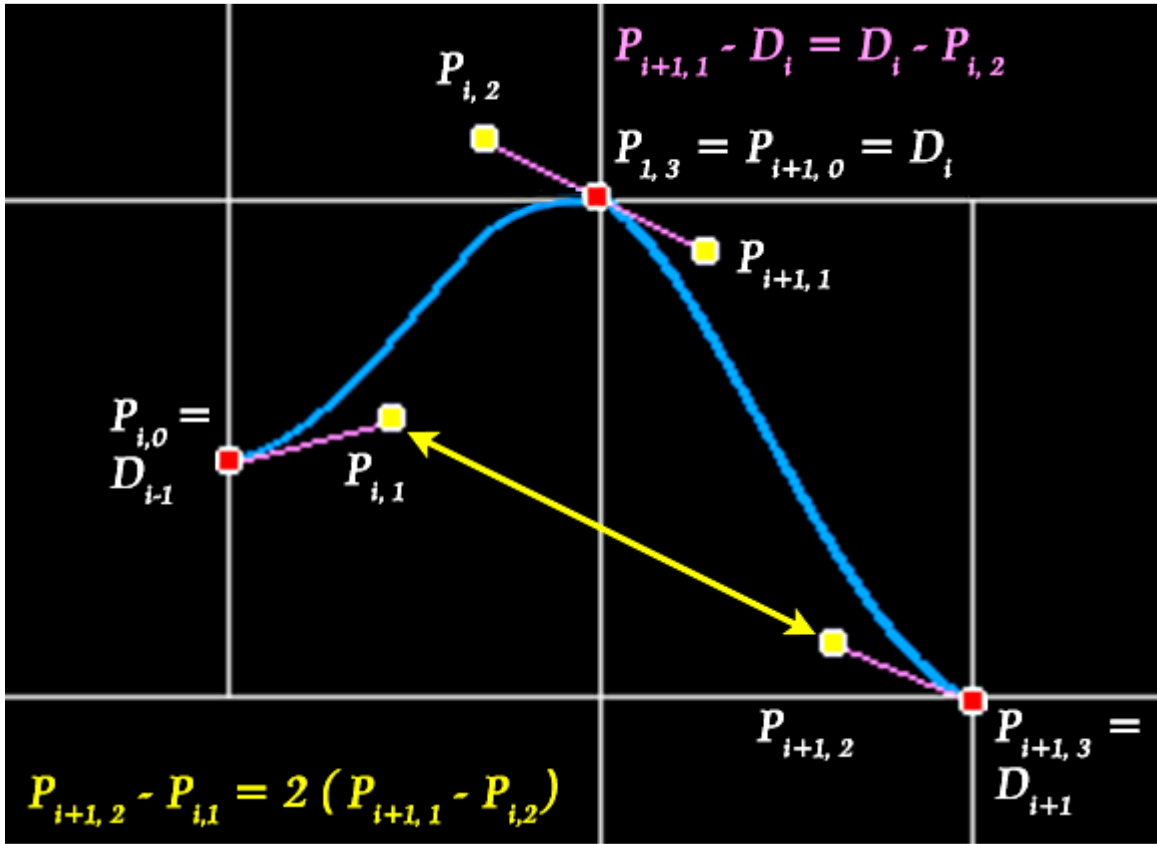
² <https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/de-Boor.html>

³ https://en.wikipedia.org/wiki/De_Boor%27s_algorithm

This is an example of a set of connected Bézier curves in which $n = 9$ data points $D_0 - D_{n-1}$ are selected. $2 * (n - 1) = 16$ additional control points $P_{i,1}$ and $P_{i,2}$ for $0 \leq i < n$ need to be calculated in order to generate a Bézier curve that connects all the data points. The data points themselves become $P_{i,0}$ and $P_{i,3}$.



For n^{th} degree curves, n conditions of continuity, designated C0 to C(n-1), can be established for the connected curve. C0 continuity simply means that the end point of one piecewise segment becomes the start point of the next. C1 continuity means that C0 continuity is present and that the first derivatives of each segment at the junction point are equal (the direction and magnitude of the tangent vectors are identical). As stated on page 1, this ensures a smoothness at the junctions without kinks. But vector drawing programs only impose G1 continuity, in which the magnitude of the vectors can differ, allowing for different “handle” lengths, and therefore more subtle adjustments to the curves. C2 continuity means that C1 continuity is present and that the second derivatives of each segment at the junction points are equal. Thus the curvatures at the junction points are continuous. Given a set of data points to be connected by a curve, a solution for the control points can be obtained, making use of the first and second derivatives, so that all 3 types of continuity are achieved. For higher-degree curves, continuities beyond C2 are possible but are quite esoteric.



A closer look shows that from the magenta equation, $P_{i,2} + P_{i+1,1} = 2D_i$

and from the yellow equation, $P_{i,1} - 2P_{i,2} + 2P_{i+1,1} - P_{i+1,2} = 0$

Finally, if the second derivatives at D_0 and D_{n-1} are set to 0:

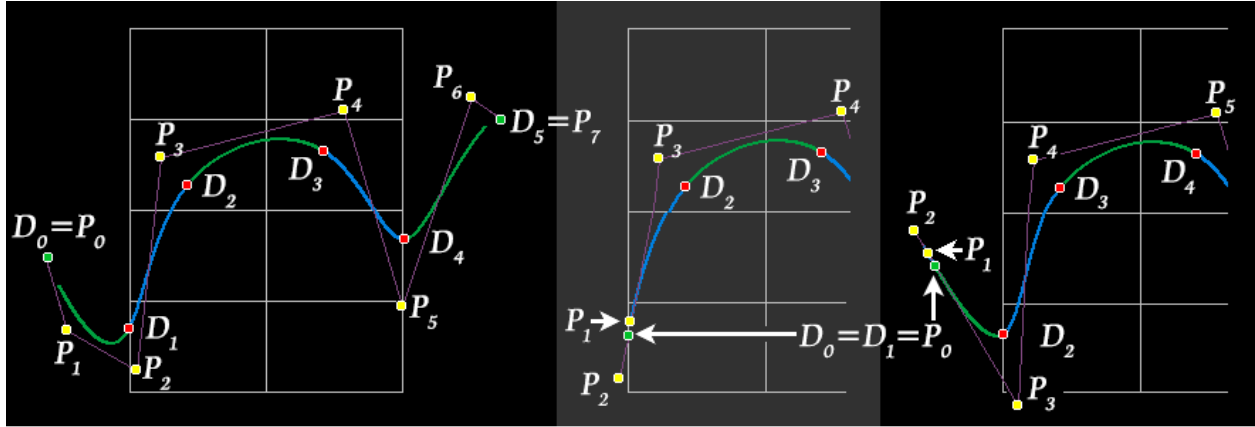
$$6(P_{0,2} - 2P_{0,1} + P_{0,0}) = 0 \longrightarrow 2P_{0,1} - P_{0,2} = D_0$$

$$6(P_{n-1,3} - 2P_{n-1,2} + P_{n-1,1}) = 0 \longrightarrow -P_{n-1,1} + 2P_{n-1,2} = D_{n-1}$$

These equations can be placed into a $[2 * (n - 1)] * [2 * (n - 1)]$ matrix which can be solved for all the necessary $2 * (n - 1)$ control points using Gaussian elimination:

$$\begin{bmatrix} 2 & -1 & & & & & \\ & 1 & 1 & & & & \\ 1 & -2 & 2 & -1 & & & \\ & & 1 & 1 & & & \\ & & 1 & -2 & 2 & -1 & \\ & & \dots & \dots & \dots & \dots & \\ & & & & 1 & 1 & \\ & & & & 1 & -2 & 2 & -1 \\ & & & & & & -1 & 2 \end{bmatrix} * \begin{bmatrix} P_{0,1} \\ P_{0,2} \\ P_{1,1} \\ P_{1,2} \\ P_{2,1} \\ \dots \\ P_{n-2,2} \\ P_{n-1,1} \\ P_{n-1,2} \end{bmatrix} = \begin{bmatrix} D_0 \\ 2D_1 \\ 0 \\ 2D_2 \\ 0 \\ \dots \\ 2D_{n-2} \\ 0 \\ D_{n-1} \end{bmatrix}$$

In order to connect 4 data points $D_1 - D_4$ with a B-spline curve (left diagram), $n = 6$ data points, including D_0 and D_5 , are required, and $n + 2 = 8$ control points $P_0 - P_7$ must be calculated. To optimize the curve, D_0 (and therefore P_0) can be set equal to D_1 (middle diagram), or an additional data point D_0 can be added and set equal to the new D_1 , analogous to the process that was described on page 3. The points to be connected graphically would then become $D_2 - D_5$. Of course the same adjustments would be added to the end of the curve as well, resulting in 8 data points and 10 control points (right diagram).



There are $n + 2$ control points, but it is known that $P_0 = D_0$ and $P_{n+1} = D_{n-1}$. So a matrix with dimensions $n * n$ must be created to solve for $P_1 - P_n$. This can be done with the basis function $N_{i,k}$ described on page 2, using deBoor's algorithm to obtain the coefficients. (See the Appendix.) The sum of each coefficient triplet is 1.

$$\begin{aligned}
 \frac{1}{4}P_1 + \frac{7}{12}P_2 + \frac{1}{6}P_3 &= D_1 \\
 \frac{1}{6}P_2 + \frac{2}{3}P_3 + \frac{1}{6}P_4 &= D_2 \\
 &\dots\dots\dots \\
 \frac{1}{6}P_{n-3} + \frac{2}{3}P_{n-2} + \frac{1}{6}P_{n-1} &= D_{n-3} \\
 \frac{1}{6}P_{n-2} + \frac{7}{12}P_{n-1} + \frac{1}{4}P_n &= D_{n-2}
 \end{aligned}$$

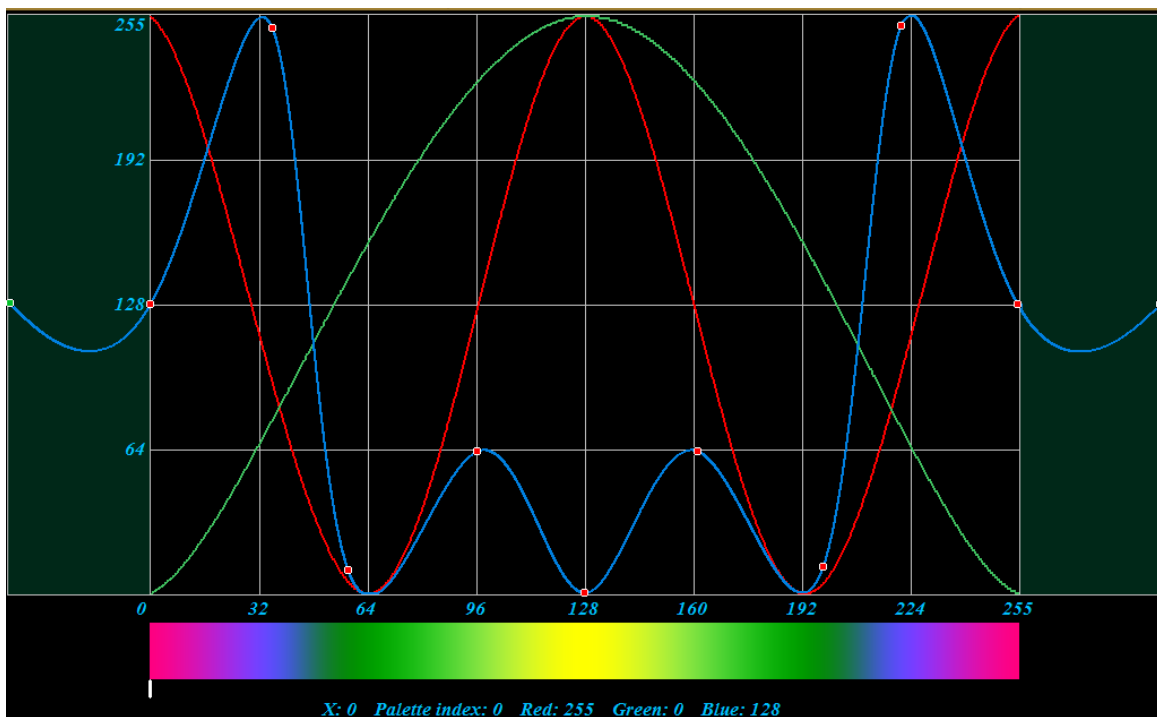
The final 2 entries, for D_0 and D_{n-1} , are obtained by setting the second derivatives at those points to 0, and substituting D_0 for P_0 and D_{n-1} for P_{n+1} .

$$\begin{aligned}
 3(2P_0 - 3P_1 + P_2) &= 0 \longrightarrow 3P_1 - P_2 = 2D_0 \\
 3(P_{n-1} - 3P_n + 2P_{n+1}) &= 0 \longrightarrow -P_{n-1} + 3P_n = 2D_{n-1}
 \end{aligned}$$

The elimination matrix is:

$$\begin{bmatrix} 3 & -1 & & & & & \\ 1/4 & 7/12 & 1/6 & & & & \\ & 1/6 & 2/3 & 1/6 & & & \\ & & & \cdots & \cdots & \cdots & \\ & & & & 1/6 & 2/3 & 1/6 \\ & & & & & 1/6 & 7/12 & 1/4 \\ & & & & & & -1 & 3 \end{bmatrix} * \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \cdots \\ P_{n-2} \\ P_{n-1} \\ P_n \end{bmatrix} = \begin{bmatrix} 2D_0 \\ D_1 \\ D_2 \\ \cdots \\ D_{n-3} \\ D_{n-2} \\ 2D_{n-1} \end{bmatrix}$$

One application of interpolated B-spline curves is the design of color palettes with smooth gradients that can be used in graphics programs. In this example, curves for red, green and blue are manipulated by dragging the data points with the mouse to create palettes of 256 entries. Each entry contains three 8-bit RGB color values which vary from 0 to 255. Palettes with 256 entries are commonly used, but any other number is equally valid. For example, an 8-bit HSV palette containing all possible hues at full saturation and value would have 1440 entries.



© Michael Sargent
 January 2020
 msargent@uvm.edu
 All graphic images were created by the author.

Appendix – the deBoor Algorithm

The deBoor algorithm offers 2 advantages in computational efficiency: 1) Only one loop from $t = 0$ to $t = 1$ is required, instead of a loop for each quartet of control points. 2) It does not compute terms which are destined to evaluate to zero.

In this example, a curve of degree 3 with $n = 7$ control points will be examined. It will therefore have $3 + 7 + 1 = 11$ knots, with 10 intervals indexed by k . The knot vector is padded at each end with three 0's and three 4's:

$$[0 \ 0 \ 0 \ 0 \ 1 \ 2 \ 3 \ 4 \ 4 \ 4 \ 4]$$

The region of the vector to be used in calculations is the middle: 0,1,2,3,4 (knot[3] $\rightarrow$ knot[7]). Here, each knot interval is 1. Since the intervals are equal, this is considered a “uniform” spline, but with additional “collapsed” starting and ending knots. This allows the final curve to start and finish on control points without gaps.

Before proceeding, the parameter t (whose range is 0 to 1) must be scaled and translated. It is located between knot[3] = 0 and knot[n] = 4. Its value is multiplied by knot[n] – knot[3] = 4.

As seen on page 2, B-splines are generated using recursive basis functions:

$$N_{i,k}(t) = \left(\frac{t - \text{knot}_i}{\text{knot}_{(i+k-1)} - \text{knot}_i} \right) \cdot N_{i,k-1}(t) + \left(\frac{\text{knot}_{(i+k)} - t}{\text{knot}_{(i+k)} - \text{knot}_{(i+1)}} \right) \cdot N_{i+1,k-1}(t)$$

Cox and deBoor found that curve points could be calculated by recursively evaluating temporary control points $d_{i,k}$ over the sections of the curve between knot[i] and knot[$i + 1$]. This process often is represented by a triangular diagram:

$$\begin{array}{ccccccc}
 d_3^3 & \xrightarrow{+ \frac{\alpha_3^3 \times \dots}{(1-\alpha_3^3) \times \dots}} & d_3^2 & \xrightarrow{+ \frac{\alpha_3^2 \times \dots}{(1-\alpha_3^2) \times \dots}} & d_3^1 & \xrightarrow{+ \frac{\alpha_3^1 \times \dots}{(1-\alpha_3^1) \times \dots}} & d_3^0 (= 0 \text{ or } 1) \\
 & \searrow & & \searrow & & \searrow & \\
 & & d_2^2 & \xrightarrow{+ \frac{\alpha_2^2 \times \dots}{(1-\alpha_2^2) \times \dots}} & d_2^1 & \xrightarrow{+ \frac{\alpha_2^1 \times \dots}{(1-\alpha_2^1) \times \dots}} & d_2^0 (= 0 \text{ or } 1) \\
 & & & \searrow & & \searrow & \\
 & & & & d_1^1 & \xrightarrow{+ \frac{\alpha_1^1 \times \dots}{(1-\alpha_1^1) \times \dots}} & d_1^0 (= 0 \text{ or } 1) \\
 & & & & & \searrow & \\
 & & & & & & d_0^0 (= 0)
 \end{array}$$

The resulting algorithm, using d and α , can be represented in optimized form as follows:

For $j = (0 \rightarrow \text{degree})$, temporary control points d_j are set to the control points $c_{j+k-\text{degree}}$, where k is the index of the knot interval that contains T , the translated and scaled value of t .

Then, for $r = 1 \rightarrow \text{degree}$:

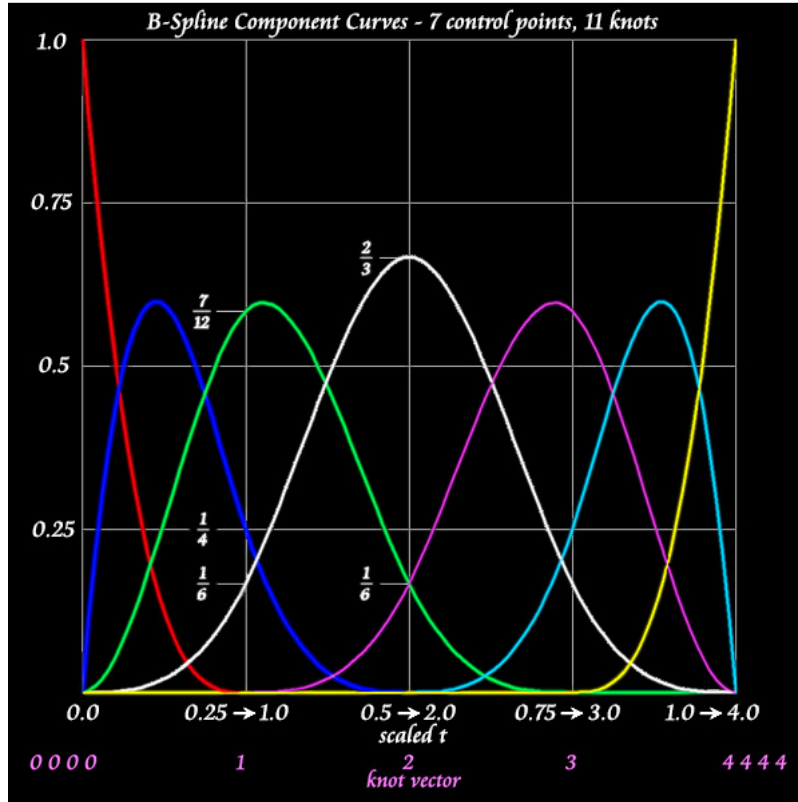
for $j = \text{degree} \rightarrow r$ (with j being decremented):

$$\alpha = \frac{T - \text{knot}_{j+k-\text{degree}}}{\text{knot}_{j+1+k-r} - \text{knot}_{j+k-\text{degree}}}$$

$$d_j = \alpha * d_j + (1 - \alpha) * d_{j-1}$$

At the completion of iteration, the final result is d_{degree} . To plot points of the B-spline curve, the algorithm must be performed twice, using both the x and y values of each control point.

Component curves for each control point can be plotted. The y values of successive control points are set to 1, with the others set to 0. The scaled t values are plotted against the y values using the algorithm. At each knot, no more than 3 curves have non-zero values. The 3 coefficients for the elimination matrix can be obtained from the curves at the knot positions, or simply by calculating them using scaled t values set to match the series of relevant knots.



Sample code for the deBoor algorithm in C is listed on the following page.

```

//----- global variables; can be set and/or changed by user input -----
int degree = 3;
double cp[7][2];           // 7 control points with x and y values
int nk = 11;               // number of knots in the knot vector: knot[ ]
double knot[11] = {0.0, 0.0, 0.0, 0.0, 1.0, 2.0, 3.0, 4.0, 4.0, 4.0, 4.0};

int deBoor(void)           // Above variables could be local and passed to the function as parameters.
{
    int k;                 // index for spline segment
    int i, j, r;           // loop index variables
    double t, T;           // parametric parameter (0 - 1), and translated/scaled version
    double alpha;          // adjustment factor
    double low, high;      // used to set the domain
    int domain[2];         // range of knot intervals
    double d[4];           // adjusted control points
    double result[2];      // x and y values

    //----- Set range for transformed t values. -----
    domain[ 0] = degree;
    domain[ 1] = nk - 1 - degree;
    low = knot[domain[ 0] ];
    high = knot[domain[ 1] ];

    for (t = 0.0; t <= 1.0; t += suitable_increment)
    {
        //----- Remap t as T to the domain where the spline is defined, and find k (the spline -----
        //----- segment index) for the t value provided. -----
        T = t * (high - low) + low;
        for ( k = domain[ 0]; k < domain[ 1]; k++ )
            if ( T >= knot[ k] && T <= knot[ k + 1] ) break;    // k contains segment index.

        for ( i = 0; i < 2; i++ )    // Loop twice for x and y values. -----
        {
            //----- the actual deBoor algorithm -----
            for ( j = 0; j < degree + 1; j++ )
                d [ j] = cp[ j + k - degree][ i];

            for ( r = 1; r < degree + 1; r++ )
                for ( j = degree; j > r - 1; j-- )
                {
                    alpha = (T - knot[ j + k - degree] ) /
                        ( knot[ j + 1 + k - r] - knot[ j + k - degree] ) ;
                    d [ j] = alpha * d[ j] + (1.0 - alpha) * d[ j - 1] ;
                }
            result[i] = d[degree]; // x and y values
        }

        //-----
        //----- Convert result[0] and result[1] to pixel coordinates and plot, using points or lines. -----
        //-----
    }
    return 0;
}

```