

A Practical Review of Uniform B-Splines

Kristin Branson

A B-spline is a convenient form for representing complicated, smooth curves. A uniform B-spline of order k is a piecewise order k Bezier curve, and is C^{k-2} -continuous (i.e. the 0^{th} through $(k-2)^{\text{th}}$ derivatives are continuous). The form of a B-spline is in general chosen because it is easy to manipulate, not because it is the solution to an optimization problem, like smoothing splines. B-splines are particularly easy to manipulate because they have local control: parameters of the B-spline only affect a small part of the entire spline. This review includes the details of B-splines that I find relevant for practically understanding and implementing B-splines.

1 Bezier Curves

A Bezier curve of order k is defined by k control points, and is a degree $k-1$ polynomial. A Bezier curve is a smooth interpolation of these k control points. A linear Bezier curve (order 2) is a linear interpolation of its two control points, p_0 and p_1 :

$$p_{01}(s) = (1-s)p_0 + sp_1,$$

defined for $0 \leq s \leq 1$. A quadratic Bezier curve (order 3) is a linear interpolation between the linear interpolation of control points p_0 and p_1 and the linear interpolation of control points p_1 and p_2 :

$$p_{01}(s) = (1-s)p_0 + sp_1, \quad p_{11}(s) = (1-s)p_1 + sp_2$$

$$\begin{aligned} p_{02}(s) &= (1-s)p_{01}(s) + sp_{11}(s) \\ &= (1-s)[(1-s)p_0 + sp_1] + s[(1-s)p_1 + sp_2] \\ &= (1-s)^2p_0 + 2s(1-s)p_1 + s^2p_2 \end{aligned}$$

This method of interpolating between interpolations is called the *deCasteljau algorithm*, and can be generalized to any order k by iterating the equation

$$p_{ij} = (1-s)p_{i,j-1} + sp_{i+1,j-1}.$$

Iterations must be carried out in order of increasing j , from $j = 0$ to k . The base case is just the control point, $p_{i0} = p_i$. For a given j , the iterations must be carried out for $i = 0, \dots, k-j-1$, in any order.

Let's see the deCasteljau algorithm for the popular cubic Bezier curve (order 4), with control points p_0 , p_1 , p_2 , and p_3 .

The linear interpolation:

$$p_{01}(s) = (1-s)p_0 + sp_1, \quad p_{11}(s) = (1-s)p_1 + sp_2, \quad p_{21}(s) = (1-s)p_2 + sp_3$$

The quadratic interpolation:

$$\begin{aligned} p_{02}(s) &= (1-s)p_{01}(s) + sp_{11}(s) \\ &= (1-s)^2p_0 + 2s(1-s)p_1 + s^2p_2 \end{aligned}$$

Similarly,

$$p_{12}(s) = (1-s)^2p_1 + 2s(1-s)p_2 + s^2p_3.$$

The cubic interpolation:

$$\begin{aligned} p_{03}(s) &= (1-s)p_{02}(s) + sp_{12}(s) \\ &= (1-s)[(1-s)^2p_0 + 2s(1-s)p_1 + s^2p_2] + s[(1-s)^2p_1 + 2s(1-s)p_2 + s^2p_3] \\ &= (1-s)^3p_0 + 3s(1-s)^2p_1 + 3s^2(1-s)p_2 + s^3p_3 \end{aligned}$$

A Bezier curve can be expressed in terms of basis functions $b_{ij}(s)$ so that

$$p_{0,k-1}(s) = \sum_{i=0}^{k-1} b_{i,k-1}(s)p_i.$$

These basis functions are the Bernstein polynomials, defined recursively by

$$b_{ij}(s) = (1-s)b_{i,j-1}(s) + sb_{i-1,j-1}(s)$$

with base cases

$$b_{00}(s) = 1, \quad b_{ij}(s) = 0 \text{ if } i < 0 \text{ or } i > j.$$

We can verify this for the cubic Bezier curve:

$$\begin{aligned} p_{03}(s) &= b_{03}(s)p_0 + b_{13}(s)p_1 + b_{23}(s)p_2 + b_{33}(s)p_3 \\ &= [(1-s)b_{02}(s) + sb_{-1,2}(s)]p_0 + [(1-s)b_{12}(s) + sb_{02}(s)]p_1 \\ &\quad + [(1-s)b_{22}(s) + sb_{12}(s)]p_2 + [(1-s)b_{32}(s) + sb_{22}(s)]p_3 \\ &= (1-s)b_{02}(s)p_0 + [(1-s)b_{12}(s) + sb_{02}(s)]p_1 \\ &\quad + [(1-s)b_{22}(s) + sb_{12}(s)]p_2 + sb_{22}(s)p_3 \\ &= (1-s)[(1-s)b_{01}(s) + sb_{-1,1}(s)]p_0 + \{(1-s)[(1-s)b_{11}(s) + sb_{01}(s)] + s[(1-s)b_{01}(s) + sb_{-1,1}(s)]\}p_1 \\ &\quad + \{(1-s)[(1-s)b_{21}(s) + sb_{11}(s)] + s[(1-s)b_{11}(s) + sb_{01}(s)]\}p_2 + s[(1-s)b_{21}(s) + sb_{11}(s)]p_3 \\ &= (1-s)^2b_{01}(s)p_0 + [(1-s)^2b_{11}(s) + 2s(1-s)b_{01}(s)]p_1 \\ &\quad + [2s(1-s)b_{11}(s) + s^2b_{01}(s)]p_2 + s^2b_{11}(s)p_3 \\ &= (1-s)^2[(1-s)b_{00}(s) + sb_{-1,0}(s)]p_0 + \{(1-s)^2[(1-s)b_{10}(s) + sb_{00}(s)] + 2s(1-s)[(1-s)b_{00}(s) + sb_{-1,0}(s)]\}p_1 \\ &\quad + \{2s(1-s)[(1-s)b_{10}(s) + sb_{00}(s)] + s^2[(1-s)b_{00}(s) + sb_{-1,0}(s)]\}p_2 + s^2[(1-s)b_{10}(s) + sb_{00}(s)]p_3 \\ &= (1-s)^3p_0 + [s(1-s)^2 + 2s(1-s)^2]p_1 + [2s^2(1-s) + s^2(1-s)]p_2 + s^3p_3 \\ &= (1-s)^3p_0 + 3s(1-s)^2p_1 + 3s^2(1-s)p_2 + s^3p_3 \end{aligned}$$

The closed form equation for the basis functions is:

$$b_{ij}(s) = \frac{j!}{i!(j-i)!} s^i (1-s)^{j-i}.$$

This basis has some nice properties:

- Partition of unity:

$$\sum_{i=0}^j b_{ij}(s) = 1, \quad b_{ij}(s) \geq 0$$

for all $0 \leq s \leq 1$.

- Affine Invariance: any linear transformation of p_i results in a linear transformation of the Bezier curve.
- The Bezier curve is tangent to the first and last segments of the control polygon.

2 B-Splines are Piecewise Bezier Curves

A B-spline of order k is a piecewise order k Bezier curve, and is C^{k-2} -continuous. Because of the continuity requirements at each breakpoint between Bezier curve pieces, only one control point of the second Bezier curve piece is free. Thus, $L + k - 1$ control points define L curve pieces for an open spline (only L control points are needed to define L curve pieces of a closed spline). We generalize the Bezier basis functions to the B-spline basis functions:

$$B_{ij}(s) = \frac{s-i}{j-1} B_{i,j-1}(s) + \frac{i+j-s}{j} B_{i+1,j-1}(s)$$

with base cases

$$B_{i1}(s) = \begin{cases} 1 & \text{if } i \leq s < i+1 \\ 0 & \text{otherwise} \end{cases}$$

for $i = 0, \dots, L-1$. The equation for a B-spline is then

$$p_{0,k-1}(s) = \sum_{i=0}^{L-1} B_{i,k-1}(s)p_i.$$

For the special case of the cubic B-spline ($k = 4$), the basis functions are

$$B_{i3}(s) = \begin{cases} \frac{1}{6}(s-i)^3 & \text{if } i \leq s < i+1 \\ \frac{1}{6}[-3(s-i-1)^3 + 3(s-i-1)^2 + 3(s-i-1) + 1] & \text{if } i+1 \leq s < i+2 \\ \frac{1}{6}[3(s-i-2)^3 - 6(s-i-2)^2 + 4] & \text{if } i+2 \leq s < i+3 \\ \frac{1}{6}[1 - (s-i-3)^3] & \text{if } i+3 \leq s < i+4 \\ 0 & \text{otherwise} \end{cases}$$

The B-spline basis functions also have some nice properties:

- Local support: Each curve piece is only a function of the four closest control points.
- The basis functions are translates of each other:

$$B_{ik}(s) = B_{0k}(s-i).$$

- The curve is C^{k-2} continuous.
- The basis functions are a partition of unity.
- Affine invariance.

3 A Convenient Representation

Because of the local support property, we can rewrite the equation for a cubic B-spline as

$$p(s) = \frac{1}{6} [(1 - (s-i))^3 p_{i-3} + [3(s-i)^3 - 6(s-i)^2 + 4] p_{i-2} + [-3(s-i)^3 + 3(s-i)^2 + 3(s-i) + 1] p_{i-1} + (s-i)^3 p_i],$$

where $i \leq s < i+1$. A similar computation can be made for any k . This can be written in matrix notation as

$$p(s) = [1 \quad s \quad s^2 \quad s^3] \mathbf{B}_i \begin{bmatrix} p_{i-3} \\ p_{i-2} \\ p_{i-1} \\ p_i \end{bmatrix},$$

again where $i \leq s < i+1$. In this equation,

$$\mathbf{B}_i = \begin{bmatrix} -\frac{1}{6}i^3 & \frac{1}{6}(3i^3 + 3i^2 - 3i + 1) & -\frac{1}{2}i^3 - i^2 + \frac{2}{3} & \frac{1}{6}(i+1)^3 \\ \frac{1}{2}i^2 & -\frac{1}{2}(3i-1)(i+1) & \frac{1}{2}(3i^2 + 4i) & -\frac{1}{2}(i+1)^2 \\ \frac{1}{2}i & \frac{1}{2}(3i+1) & -\frac{1}{2}(3i+2) & \frac{1}{2}(i+1) \\ \frac{1}{6} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{6} \end{bmatrix}.$$

We can also include the placement matrix \mathbf{G}_i :

$$p(s) = [1 \quad s \quad s^2 \quad s^3] \mathbf{B}_i \mathbf{G}_i \mathbf{p},$$

where \mathbf{p} is the vector of control points,

$$\mathbf{p} = \begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_n \end{bmatrix}.$$

For an open spline

$$G_i[m, n] = \begin{cases} 1 & \text{if } n = i + m - 3 \\ 0 & \text{otherwise} \end{cases}$$

For a closed spline, the placement matrix is similar:

$$G_i[m, n] = \begin{cases} 1 & \text{if } n - 1 = i + m - 3 - 3 - 1 \pmod{L} \\ 0 & \text{otherwise} \end{cases}$$

This form is convenient for taking derivatives or integrals w.r.t. s , since the only term in this equation that depends on s is the first row vector. This simplifies calculations like computing the curve tangents and normals, or the curve area, centroid, and variance.

4 Fitting a B-Spline to Data Points

Suppose we want to find the set of control points \mathbf{P} that best interpolates a set of data points $\{(s_i, \mathbf{x}_i)\}_{i=1}^N$. We use best in the least-squares sense, i.e. we want to find \mathbf{P} that minimizes:

$$h[\mathbf{P}] = \sum_{i=1}^N (\mathbf{x}_i - \mathbf{p}(s_i; \mathbf{P}))^2.$$

To solve this problem, we can write the function $\mathbf{p}(s_i; \mathbf{P})$ in vector form:

$$\mathbf{P}(s; \mathbf{P}) = \mathbf{A}_s \mathbf{P}.$$

It must then be that \mathbf{A}_s is a $N \times L$ matrix such that

$$\begin{aligned} \mathbf{P}(s_i; \mathbf{P}) &= \mathbf{A}_s(i, :) \mathbf{P} \\ &= \sum_{j=0}^{L-1} B_{j, k-1}(s_i) p_j \end{aligned}$$

Thus, entry $\mathbf{A}_s(i, j) = B_{j-1, k-1}(s_i)$. Notice that each row of \mathbf{A}_s will be nonzero for at most four consecutive entries (exactly four for a closed B-spline). Given this vector notation for $\mathbf{P}(s_i; \mathbf{P})$, we can write the optimization as

$$\mathbf{P}^* = \min_{\mathbf{P}} \text{trace}[(\mathbf{X} - \mathbf{A}_s \mathbf{P})^\top (\mathbf{X} - \mathbf{A}_s \mathbf{P})].$$

To find the minimizing \mathbf{P} , we can take the derivative w.r.t. \mathbf{P} and set it to $\mathbf{0}$, then solve for \mathbf{P} . This results in

$$\mathbf{P}^* = (\mathbf{A}_s^\top \mathbf{A}_s)^{-1} \mathbf{A}_s^\top \mathbf{X}.$$

5 Matlab Implementations

I have implemented a number of B-spline functions in Matlab. To get started with B-splines, I recommend playing with the function `input_spline.m`. This function allows the user to input a closed B-spline on top of an image. The most interesting function called by `input_spline.m` is `fit_closed_b_spline.m`. This function fits a closed B-spline to the input data points, as described in Section 4. This function can be easily extended to open splines, if one wishes.

I have also implemented some basic functions for dealing with cubic, uniform B-splines. The functions `evaluate_spline.m` and `evaluate_spline_curve.m` evaluate the B-spline (1D and 2D, respectively) defined by the input control points

at the input locations. These both use the convenient representation discussed in Section 3. The \mathbf{B}_i and \mathbf{G}_i matrices discussed in section 3 are created using the function `setup_cubic_splines.m`, which sets up the matrices for the desired number of curve pieces. The functions `evaluate_spline_prime.m` and `evaluate_normal.m` compute the tangents and normals of the B-spline at the input locations, again using the representation of Section 3. The function `transform_coefs.m` takes advantage of the affine invariance of the B-spline basis functions, and applies a translation, scaling, and rotation to an input B-spline. Note that the functions described in this paragraph have an interface that allows multiple sets of control points, defining multiple B-splines, be input and evaluated at once. This is because my implementation is designed for tracking contours using particle filtering.

[]

References

- [1] S. Buss. *3-D Compute Graphics*. Cambridge University Press, New York, 2003.
- [2] E. Demidov. An interactive introduction to splines website: <http://www.ibiblio.org/e-notes/splines/intro.htm>, 2004.
- [3] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer Verlag, Basel, 2001.