Uniform B-Spline Curves

Uniform B-Spline curves are piecewise polynomial curves associated with a sequance of control points of any length. Unlike the Bezier curves, their degree is independent of the number of control points.

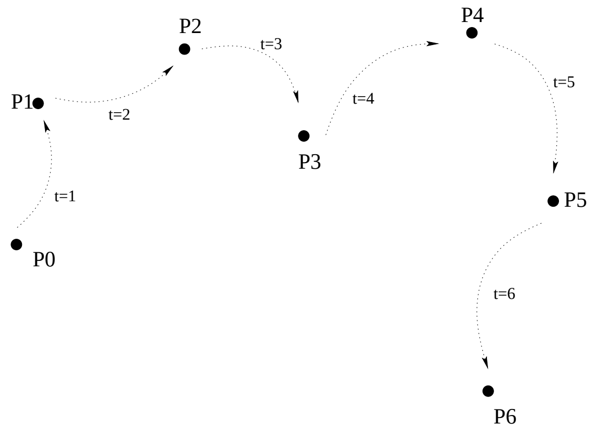

Figure 1: The 'curve' B_0.

B-Spline curves can be constructed by recursive averaging. Start from a curve which stays one second at the first control point, P_0, then jumps to the second one, P_1, then after one second to the third etc (Figure 1). This is going to be our zero-degree B-spline curve. (we'll generously call it a curve even though it doesn't really look like a curve). The formula for that curve (call it B_0) is

$$B_0(t) = P_i \text{ for } t \in [i, i+1].$$

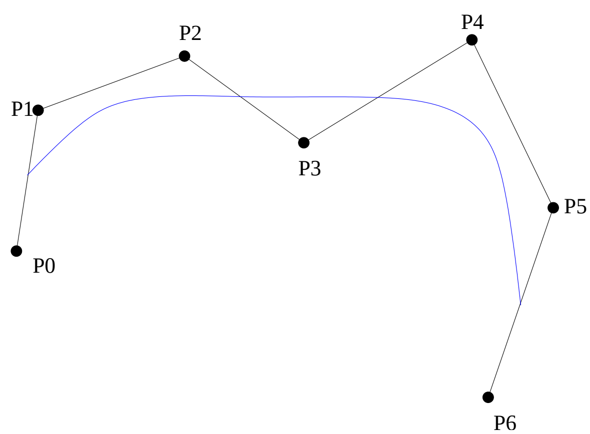

Figure 2: B_0 (black) and B_1 (blue).

Now we will start making the curve more regular by averaging/filtering it. We define a new curve (degree-one B-spline) B_1 by setting $B_1(t)$ to the average position of a point moving along the curve B_0 over time interval $[t, t+1]$. Thus,

$$B_1(t) = \int_t^{t+1} B_0(s)ds.$$

Now, it's not hard to see that B_1 is a polygonal curve joining the control points (Figure 2. It's not differentiable, but it is definitely nicer-looking than B_0. The averaging procedure can be applied to B_1 to lead to B_2 (degree-two B-spline) and so on. Thus, we have a recursive formula

$$B_{i+1}(t) = \int_t^{t+1} B_i(s)ds.$$

1 Smoothness

By smoothness we mean the number of derivatives that can be computed for a curve. The more derivatives the smoother the curve. We'll say that a curve B is C^k if the derivarives $B'(t), B''(t) \ldots B^{(k)}(t)$ exist for all t in the parameter range, i.e. everywhere where B is defined, and they are all continuous. We'll say a curve is C^0 if it's continuous.

We can now say that B_0, the zero-degree B-spline is not even C^0. The next curve in the hierarchy, B_1, is continuous but not differentiable (it has sharp corners at the control points, certainly no derivative there). Hence it's C^0 but not C^1. It turns out that B_2 is a C^1 curve. This is a consequence of the Fundamental Theorem of Calculus, which allows to compute

$$\frac{d}{dt}B_2(t) = B_1(t+1) - B_1(t),$$

which means that the derivative can be expressed using B_1 and hence is continuous since B_1 is. By applying this argument once again, we can show that the degree-3 B-splines (also called cubic B-splines) are C^2, i.e. have continuous second derivative. In general, B_k is C^{k-1}.

In applications, the cubic B-spline curves are the most common: they seem to hit the right spot in the tradeoff between quality (smoothness) and simplicity (low degree). They are also well-suited for modeling mased on physics. Imagine traveling in a spaceship moving along the curve B_k. For B_0, it's even hard to imagine. For B_1 this is also problematic, since it would require infinite acceleration at the control points; this would make the trip bumpy and unpleasant. B_2 seems more plausible, at least not exposing us to infinite accelerations. However, it would not be fun at the points where the acceleration changes abruptly (imagine that you have nothing to hold to and at a certain moment the acceleration (generating a force on you because of the innertia of your body) changes abruptly: you probably won't have time to adjust yourself and would fall down). Eventually, B_3 seems to be great, offering continuous change of the acceleration and thus a pleasant journey. This is also somehow related to how we tend to drive into a sharp turn: turning the wheel too fast can be felt immediately and even make the car unstable. Thus, we try to change the turning angle (closely related to curvature and second derivative) over a longer time, making it as slowly-varying as possible.

2 Polar form and DeBoor algorithm

B-splines are easy to specify using polar forms. If the control points are $P_0, P_1, \ldots, P_n$ then the segment of the uniform degree-n B-spline curve corresponding to parameters $t \in [i, i+1]$ is given by

$$\begin{aligned} P(i+1-n, i+3-n, \ldots, i) &= P_i \\ P(i+2-n, i+4-n, \ldots, i+1) &= P_{i+1} \\ &\ldots \\ P(i+1, i+2, \ldots, i+n) &= P_{i+n}. \end{aligned}$$

For cubic B-splines, the above equations take the form:

$$P(i-2, i-1, i) = P_i$$

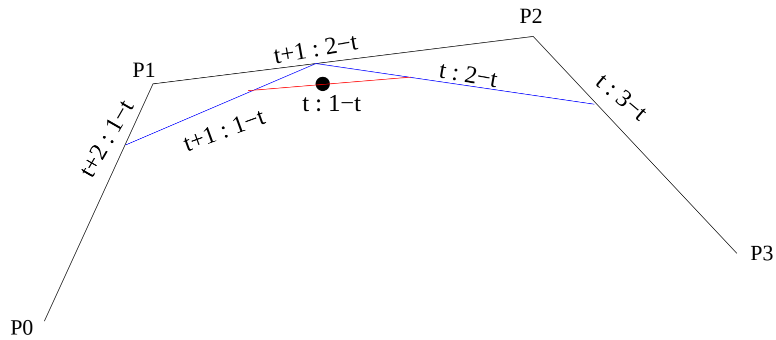

Figure 3: DeBoor algorithm for cubic B-spline curve; here we show calculation for 4 control points and parameter values in $[0,1]$.

$$
\begin{aligned}
P(i-1,i,i+1) &= P_{i+1} \\
P(i,i+1,i+2) &= P_{i+2} \\
P(i+1,i+2,i+3) &= P_{i+3}.
\end{aligned} \tag{1}
$$

One can compute $B(t) = P(t,t,t)$ using the DeBoor algorithm, which is very much alike the DeCasteljau algorithm for Bezier curves. Here is how it proceeds for cubic B-splines. First, we compute $P(i-1,i,t)$, $P(i,i+1,t)$ and $P(i+1,i+2,t)$ from equations (1):

$$
\begin{aligned}
P(i-1,i,t) = P(i-1,i,\frac{t-(i-2)}{3} * (i+1) + \frac{(i+1)-t}{3} * (i-2)) = \\
\frac{t-(i-2)}{3} P(i-1,i,i+1) + \frac{(i+1)-t}{3} P(i-1,i,i-2) = \\
\frac{t-(i-2)}{3} P_{i+1} + \frac{(i+1)-t}{3} P_i
\end{aligned}
$$

and similarly

$$
\begin{aligned}
P(i,i+1,t) &= \frac{t-(i-1)}{3} P_{i+2} + \frac{(i+2)-t}{3} P_{i+1} \\
P(i+1,i+2,t) &= \frac{t-i}{3} P_{i+3} + \frac{(i+3)-t}{3} P_{i+2}
\end{aligned}
$$

Now, we compute $P(i,t,t)$ and $P(i+1,t,t)$:

$$
P(i,t,t) = P(i,t,\frac{t-(i-1)}{2} * (i+1) + \frac{i+1-t}{2} * (i-1)) = \frac{t-(i-1)}{2} P(i,i+1,t) + \frac{i+1-t}{2} P(i-1,i,t)
$$

and

$$
P(i+1,t,t) = P(i,t,\frac{t-i}{2} * (i+2) + \frac{i+2-t}{2} * i) = \frac{t-i}{2} P(i+1,i+2,t) + \frac{i+2-t}{2} P(i,i+1,t).
$$

Finally, we can compute $P(t,t,t)$ as follows:

$$
P(t,t,t) = P(t,t,(i+1-t) * i + (t-i) * (i+1)) = (i+1-t) P(t,t,i) + (t-i) P(t,t,i+1).
$$

Notice that the overall scheme of this algorithm is the same as that of the DeCasteljau algorihtm: we compute weighted averages of pairs of control points, then weighted averages of consecutive pairs of resulting points etc. However, the weights in DeBoor algorithm are different from weights in the DeCasteljau algorithm. Geometric interpretation of this process is shown in Figure 3.