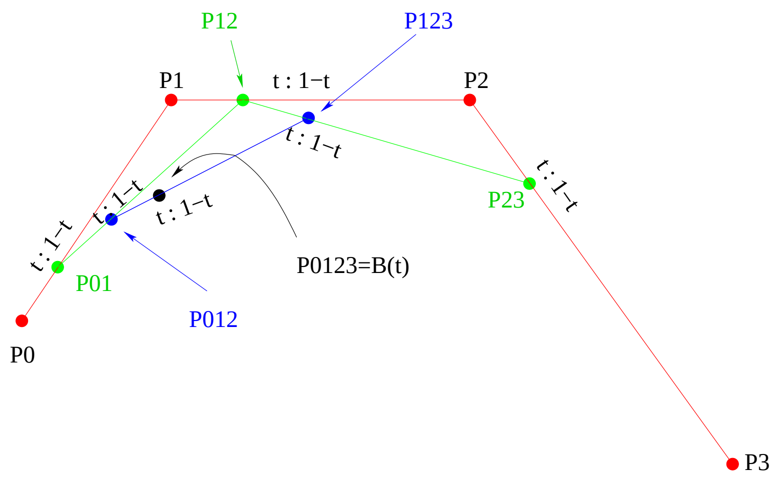

Figure 1: De Casteljau algorithm. The control points P_0, P_1, P_2 and P_3 are joined with line segments (shown in red; those red intervals form something called 'control polygon', even though they are not really a polygon but rather a polygonal curve). Each of them is then divided in the same ratio $t : 1 - t$, giving rise to the green points P_{01}, P_{12} and P_{23}. Again, each consecutive two green points are joined with line segments which are subdivided with blue points P_{012} and P_{123}. After the interval joining the blue points is subdivided, only one point is left (here: the black one). This is the location of our moving point at time t. The trajectory of that point for times between 0 and 1 is the Bezier curve. Of course, the more control points the more colors I would need to use, i.e. the more the 'levels' of subdivision are needed.

Bezier Curves

Curves are trajectories of moving points. We'll specify them as functions assigning a location of that moving point (in 2D or 3D) to a parameter t (think of it as time). In general, all curves we are going to talk about will be defined by a sequence of control points. Curves useful in in geometric modeling should have shape which has a clear and intuitive relation to the path of the sequence of control points. One family of curves satisfying this requirement are Bezier Curves.

1 Definition and a few formulas and properties

A Bezier curve is defined by a sequence of $N + 1$ control points, P_0, P_1, $\ldots$, P_N. We defined the Bezier curve using the de Casteljau algorithm, based on recursive splitting of the intervals joining the consecutive control points (Figure 1). It should be clear from the picture that $B_{P_0 P_1 \ldots P_N}(0) = P_0$ and $B_{P_0 P_1 \ldots P_N}(1) = P_N$, i.e. the curve starts at the first and ends at the last control point (think what it means to split in 1:0 or 0:1 ratio...). $\vec{P_0 P_1}$ and $\vec{P_{N-1} P_N}$ are tangent vectors to the curve at its endpoints (Figure 2; also easily visible in the unrelated Figure 3). The latter will be straightforward to verify once we have an explicit formula for the Bezier curve at the end of this section. In general, if $P_0, P_1, \ldots, P_n$ are control points for the curve then $B'(0) = n\vec{P_0 P_1}$ and $B'(1) = n\vec{P_{n-1} P_n}$.

Notice that the endpoints of the interval which is split as the last one (the blue one in Figure 1) are points of Bezier curves defined by sequences of control points $P_0 P_1 \ldots P_{N-1}$ and $P_1 P_2 \ldots P_N$. This leads to the formula

$$B_{P_0 P_1 \ldots P_N}(t) = (1 - t) * B_{P_0 P_1 \ldots P_{N-1}}(t) + t * B_{P_1 P_2 \ldots P_N}(t).$$

Remembering that for just one control point we have constant Bezier curve i.e. $B_{P_0}(t) = P_0$, we can use

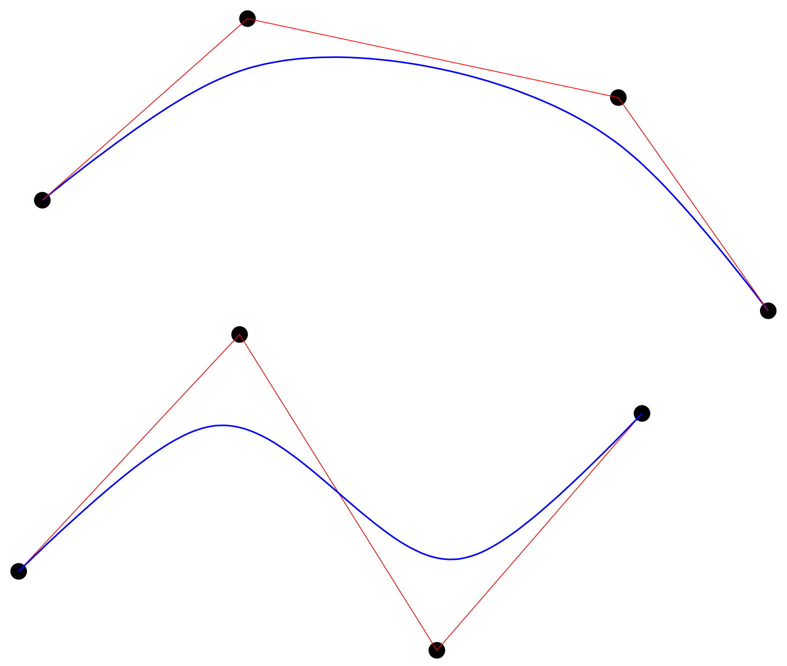

Figure 2: Bezier curves; black dots are the control points and the curve is shown in blue. Both curves have 4 control points and therefore are given by cubic polynomials. Notice that the interval joining the first two control points is tangent to the curve at its starting point and the line joining the last two control points is tangent to the curve at its final point.

the above equation to derive an explicit formula for a B-spline curve of any degree. Here's how:

$$B_{P_0P_1} = (1-t) * B_{P_0} + t * B_{P_1} = (1-t) * P_0 + t * P_1$$

having this:

$$\begin{aligned} B_{P_0P_1P_2} &= (1-t) * B_{P_0P_1} + t * B_{P_1P_2} = \\ &= (1-t) * ((1-t) * P_0 + t * P_1) + t * ((1-t) * P_1 + t * P_2) = \\ &= (1-t)^2 * P_0 + 2t(1-t) * P_1 + t^2 * P_2. \end{aligned}$$

It's not hard to guess that, in general, to evaluate $B_{P_0P_1\ldots P_N}$ we need to weight the consecutive control points with the terms of the expansion of $((1-t)+t)^{N-1}$. That is, the weight sequences are (see above) 1 (for $N=0$); $(1-t), t$ ($N=1$); $(1-t)^2, 2t(1-t), t^2$ ($N=2$); and $(1-t)^3$, $3(1-t)^2t$, $3(1-t)t^2$, t^3 for $N=3$.

The weights (as functions of t) for a fixed N are called *Bezier basis functions of degree* N. Note that this is indeed a basis (in the linear algebra sense) for the space of all polynomials: in fact, every polynomial curve can be expressed as a Bezier curve for a certain control point. Thus, Bezier curves of degree d are exactly polynomial curves of degree d. This might seem a bit disappointing at first, but remember that what we gain is an intuitive way to control the curve's shape based on control points. Check out one of the applets to experience this.

If we have 5 control points, then the weights are $(1-t)^4, 4t(1-t)^3, 6t^2(1-t)^2, 4t^3(1-t), t^4$. Thus, the formula for the B-spline curve (which is a degree-4 polynomial in this case) is

$$B_{P_0P_1P_2P_3P_4} = (1-t)^4 * P_0 + 4t(1-t)^3 * P_1 + 6t^2(1-t)^2 * P_2 + 4t^3(1-t) * P_3 + t^4 * P_4.$$

Notice that this means that each point on a Bezier curve is a weighted average of the input points.

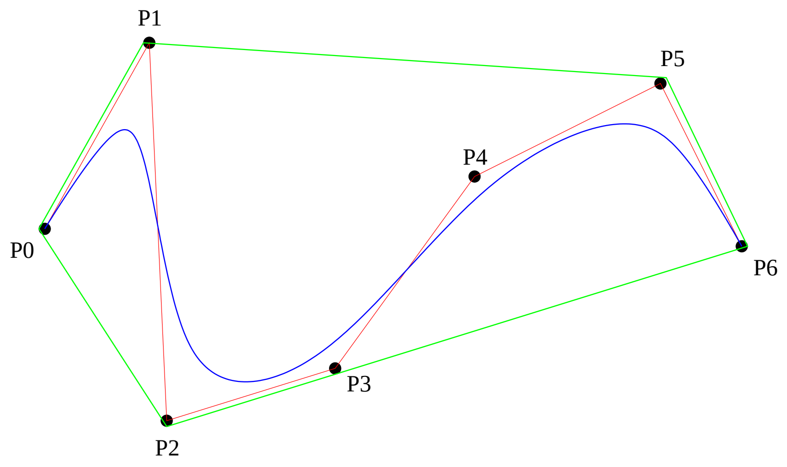

Figure 3: The control polygon is shown red. The convex hull of the control points is the green polygon (smallest convex having all the control points inside). The Bezier curve is shown in blue. The convex hull property states that blue has to be contained inside the green polygon.

Therefore, it lies inside the *convex hull* of the control points, i.e. the smallest convex polygon enclosing them (see Figure 3).

2 Joining Bezier segments

Based on the previous section, it is not hard to figure out how to join two or more Bezier curves so that they all form a smooth curve. Figure 4 explains it all.

3 Polar representations of polynomial curves

Polar forms provide a very powerful formalism for defining as well as manipulating and understanding polynomial curves. In this section we discuss the basics of this approach.

3.1 Definition

Take any polynomial of one variable P and let d be an integer such that deg $P \leq d$. Then, P can be written in a unique way in the following form:

$$P(t) = M(t, t, \dots, t), \tag{1}$$

where M is a *symmetric* and *multiaffine* function of d variables.

Symmetry means that permuting arguments does not change the value of M. Formally,

$$M(x_1, x_2, \dots, x_d) = M(x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(d)})$$

for any permutation π of $\{1, 2, \dots, d\}$. For example, if $d = 3$, $M(1,1,0) = M(0,1,1) = M(1,0,1)$. If an argument x of M is repeated k times, instead of writing it k times, we'll just write x^k. For example, $M(0,0,1) = M(0^2,1) = M(0^2,1^1)$.

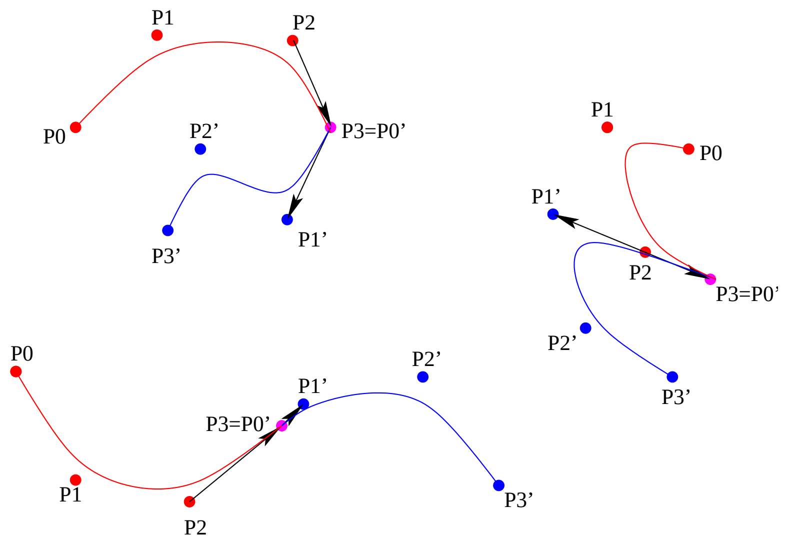

Figure 4: Joining two Bezier curves. Control points for one are red and for the other (primed) – blue. We want them to *join*, so the second one should start where the first ends. Thus, the last control point of the first curve has to overlap with the first of the second curve. This point is shown in magenta (=red+blue). Top left: the vectors $\vec{P_2P_3}$ and $\vec{P_0'P_1'}$ are not collinear. Therefore, the red and blue curves have different tangent lines at the point they meet at (magenta dot) and so they don't join smoothly there. Right: the vectors are collinear now, but they point in different directions. So, the red curve arrives from one direction, at the magenta point, but the blue one leaves in the opposite direction. This causes a very ('infinitely?') sharp cusp in the curve. Bottom left: The vectors are parallel and they point in the same direction; the two curves join smoothly.

Before we say what multiaffine means, let's define what affine means for a function. Basically, a function $A : V \to W$ (V,W are vector spaces) is affine if it preserves weighted averages:

$$A(\alpha x + (1-\alpha)y) = \alpha A(x) + (1-\alpha)A(y)$$

for any x, y in the domain and any scalar α.

A function of several variables is multiaffine if it is affine with respect to every variable. Since our M has to be symmetric (therefore we are free to permute the variables without changing the value), it is enough to say that it has to be affine with respect to the first variable:

$$M(\alpha x + (1-\alpha)y, x_2, x_3, \dots, x_d) = \alpha M(x, x_2, x_3, \dots, x_d) + (1-\alpha)M(y, x_2, x_3, \dots, x_d).$$

In what follows, we'll call M a *polar form* or a *blossom* of P.

3.2 A few examples or polar forms

Example 1. Let $P(t) = 1 + x + 2x^2 + x^3$ and d=3. Then, the polar form of P is given by

$$M(x, y, z) = 1 + \frac{1}{3}(x + y + z) + \frac{2}{3}(xy + xz + yz) + xyz.$$

The equation $P(t) = M(t, t, t)$ is straightforward to check. Symmetry is also easy to see (swapping any two variables doesn't change the formula for M). Multiaffinity is also not hard to see either: imagine you fix two variables, say, y and z, making them constants. Then, the formula for M reduces to a degree-1 polynomial in x:

$$\Big(1+\frac{1}{3}(y+z)+\frac{2}{3}yz\Big)+\Big(\frac{1}{3}+\frac{2}{3}(y+z)+yz\Big)*x.$$

Degree-1 polynomials are affine functions and therefore P is multiaffine.

Example 2. For the same polynomial P and $d=5$, the polar form is

$$M(x,y,z,t,u)=1+\frac{1}{5}(x+y+z+t+u)+\frac{2}{10}(xy+xz+xt+xu+yz+yt+yu+zt+zu+tu)+$$
$$+\frac{1}{10}(xyz+xyt+xyu+xzt+xzu+xtu+yzt+yzu+ytu+ztu).$$

It is different from the polar form obtained in the previous example, but this does not contradict the uniqueness of the polar form (since this one is based on a different d).

Conclusion. The general procedure for writing the polar form is to define it as a linear combination of symmetric polynomials of d variables. The symmetric polynomials of degree k and of d variables is simply a sum of all possible products of k different variables. These expressions can get long fast, but good news is that, in practice, one doesn't really need them.

3.3 Bezier curve blossoms

One reason that makes polar forms useful is that by putting constraints on them we can define polynomials. For many polynomial curves common in CAD/CAM systems the polar forms are particularly simple.

Bezier curve of degree d can, for example, be defined by the following constraints. Let $P_0, P_1, \ldots, P_d$ be its control points. Bezier curve has polar form that satisfies

$$M(0^{n-k}1^k)=P_k$$

for $k=0,1,2,\ldots,d$. In fact, the de Casteljau algorithm can be viewed as a procedure for computing $P(t)=M(t,t,\ldots,t)$. Below we elaborate on this relationship for cubic B-splines. Same argument works for Bezier curves of all other degrees.

In the case of cubic B-splines, the polar form is defined by

$$\begin{aligned}M(0,0,0)&=P_0\\ M(0,0,1)&=P_1\\ M(0,1,1)&=P_2\\ M(1,1,1)&=P_3.\end{aligned}\tag{2}$$

We want to compute $P(t)=M(t,t,t)$. Here is a way to do it. We use the constraints on M to compute $M(0,0,t)$, $M(0,1,t)$, $M(1,1,t)$:

$$\begin{aligned}M(0,0,t)=M(0,0,t*1+(1-t)*0)&=tM(0,0,1)+(1-t)M(0,0,0)=\\ &=tP_1+(1-t)P_0(=P_{01})\\ M(0,1,t)=M(0,t,1)=M(0,t*1+(1-t)*0,1)&=tM(0,1,1)+(1-t)M(0,0,1)=\\ &=tP_2+(1-t)P_1(=P_{12})\\ M(1,1,t)=M(t,1,1)=M(t*1+(1-t)*0,1,1)&=tM(1,1,1)+(1-t)M(0,1,1)=\\ &=tP_3+(1-t)P_2(=P_{23}).\end{aligned}$$

Points with multiple subscripts are the same as in the de Casteljau algorithm run for parameter t. Using the above we compute $M(0,t,t)$ and $M(1,t,t)$:

$$\begin{aligned}
M(0,t,t) = M(0, t*1 + (1-t)*0, t) = tM(0,1,t) + (1-t)M(0,0,t) = \\
= tP_{12} + (1-t)P_{01} (= P_{012}) \\
M(1,t,t) = M(1, t*1 + (1-t)*0, t) = tM(1,1,t) + (1-t)M(1,0,t) = \\
= tP_{23} + (1-t)P_{12} (= P_{123}).
\end{aligned}$$

Finally, we are done:

$$\begin{aligned}
P(t) = M(t,t,t) = M(t*1 + (1-t)*0, t, t) = tM(1,t,t) + (1-t)M(0,t,t) = \\
= tP_{123} + (1-t)P_{012} (= P_{0123}).
\end{aligned}$$

This can be considered an algebraic interpretation of the de Casteljau algorithm: it's actually equivalent to computing a point on the Bezier curve from a polar representation constrained by Equations 2.