The de Casteljau Algorithm

4

The algorithm described in this chapter is probably the most fundamental one in the field of curve and surface design, yet it is surprisingly simple. Its main attraction is the beautiful interplay between geometry and algebra: a very intuitive geometric construction leads to a powerful theory.

Historically, it is with this algorithm that the work of de Casteljau started in 1959. The only written evidence is in [145] and [146], both technical reports that are not easily accessible. De Casteljau's work went unnoticed until W. Boehm obtained copies of the reports in 1975. Since then, de Casteljau's work has gained more popularity.

4.1 Parabolas

We give a simple construction for the generation of a parabola; the straightforward generalization will then lead to Bézier curves. Let $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$ be any three points in $\mathbb{E}^3$, and let $t \in \mathbb{R}$. Construct

$$\mathbf{b}_0^1(t) = (1-t)\mathbf{b}_0 + t\mathbf{b}_1,$$

$$\mathbf{b}_1^1(t) = (1-t)\mathbf{b}_1 + t\mathbf{b}_2,$$

$$\mathbf{b}_0^2(t) = (1-t)\mathbf{b}_0^1(t) + t\mathbf{b}_1^1(t).$$

Inserting the first two equations into the third one, we obtain

$$\mathbf{b}_0^2(t) = (1-t)^2\mathbf{b}_0 + 2t(1-t)\mathbf{b}_1 + t^2\mathbf{b}_2. \tag{4.1}$$

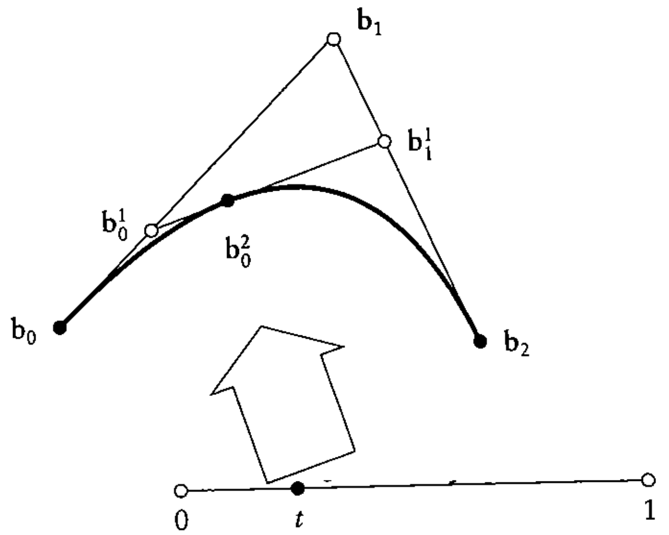

Figure 4.1 Parabolas: construction by repeated linear interpolation.

This is a quadratic expression in t (the superscript denotes the degree), and so $\mathbf{b}_0^2(t)$ traces out a *parabola* as t varies from $-\infty$ to $+\infty$. We denote this parabola by $\mathbf{b}^2$. This construction consists of *repeated linear interpolation*; its geometry is illustrated in Figure 4.1. For t between 0 and 1, $\mathbf{b}^2(t)$ is inside the triangle formed by $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$; in particular, $\mathbf{b}^2(0) = \mathbf{b}_0$ and $\mathbf{b}^2(1) = \mathbf{b}_2$.

Inspecting the ratios of points in Figure 4.1, we see that

$$\text{ratio}(\mathbf{b}_0, \mathbf{b}_0^1, \mathbf{b}_1) = \text{ratio}(\mathbf{b}_1, \mathbf{b}_1^1, \mathbf{b}_2) = \text{ratio}(\mathbf{b}_0^1, \mathbf{b}_0^2, \mathbf{b}_1^1) = t/(1-t).$$

Thus our construction of a parabola is *affinely invariant* because piecewise linear interpolation is affinely invariant; see Section 3.2.

We also note that a parabola is a plane curve, since $\mathbf{b}^2(t)$ is always a barycentric combination of three points, as is clear from inspecting (4.1). A parabola is a special case of *conic sections*, which will be discussed in Chapter 12.

Finally we state a theorem from analytic geometry, closely related to our parabola construction. Let $\mathbf{a}, \mathbf{b}, \mathbf{c}$ be three distinct points on a parabola. Let the tangent at $\mathbf{b}$ intersect the tangents at $\mathbf{a}$ and $\mathbf{c}$ in $\mathbf{e}$ and $\mathbf{f}$, respectively. Let the tangents at $\mathbf{a}$ and $\mathbf{c}$ intersect in $\mathbf{d}$. Then ratio$(\mathbf{a}, \mathbf{e}, \mathbf{d}) =$ ratio$(\mathbf{e}, \mathbf{b}, \mathbf{f}) =$ ratio$(\mathbf{d}, \mathbf{f}, \mathbf{c})$. This *three tangent theorem* describes a property of parabolas; the de Casteljau algorithm can be viewed as the constructive counterpart. Figure 4.1, although using a different notation, may serve as an illustration of the theorem.

4.2 The de Casteljau Algorithm

Parabolas are plane curves. However, many applications require true space curves.[1] For those purposes, the previous construction for a parabola can be generalized to generate a polynomial curve of arbitrary degree n:

de Casteljau algorithm:

Given: $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{E}^3$ and $t \in \mathbb{R}$,

set

$$\mathbf{b}_i^r(t) = (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t) \quad \begin{cases} r = 1, \ldots, n \\ i = 0, \ldots, n-r \end{cases} \tag{4.2}$$

and $\mathbf{b}_i^0(t) = \mathbf{b}_i$. Then $\mathbf{b}_0^n(t)$ is the point with parameter value t on the *Bézier curve* $\mathbf{b}^n$, hence $\mathbf{b}^n(t) = \mathbf{b}_0^n(t)$.

The polygon $\mathbf{P}$ formed by $\mathbf{b}_0, \ldots, \mathbf{b}_n$ is called the *Bézier polygon* or *control polygon* of the curve $\mathbf{b}^n$.[2] Similarly, the polygon vertices $\mathbf{b}_i$ are called *control points* or *Bézier points*. Figure 4.2 illustrates the cubic case.

Sometimes we also write $\mathbf{b}^n(t) = B[\mathbf{b}_0, \ldots, \mathbf{b}_n; t] = B[\mathbf{P}; t]$ or, shorter, $\mathbf{b}^n = B[\mathbf{b}_0, \ldots, \mathbf{b}_n] = B\mathbf{P}$. This notation[3] defines B to be the (linear) operator that associates the Bézier curve with its control polygon. We say that the curve $B[\mathbf{b}_0, \ldots, \mathbf{b}_n]$ is the *Bernstein–Bézier approximation* to the control polygon, a terminology borrowed from approximation theory; see also Section 6.9.

The intermediate coefficients $\mathbf{b}_i^r(t)$ are conveniently written into a triangular array of points, the *de Casteljau scheme*. We give the example of the cubic case:

$$\begin{array}{llll} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0^1 & & \\ \mathbf{b}_2 & \mathbf{b}_1^1 & \mathbf{b}_0^2 & \\ \mathbf{b}_3 & \mathbf{b}_2^1 & \mathbf{b}_1^2 & \mathbf{b}_0^3. \end{array} \tag{4.3}$$

This triangular array of points seems to suggest the use of a two-dimensional array in writing code for the de Casteljau algorithm. That would be a waste of

1 Compare the comments by P. Bézier in Chapter 1!

2 In the cubic case, there are four control points; they form a tetrahedron in the 3D case. This tetrahedron was already mentioned by W. Blaschke [65] in 1923; he called it "osculating tetrahedron."

3 This notation should not be confused with the blossoming notation used later.

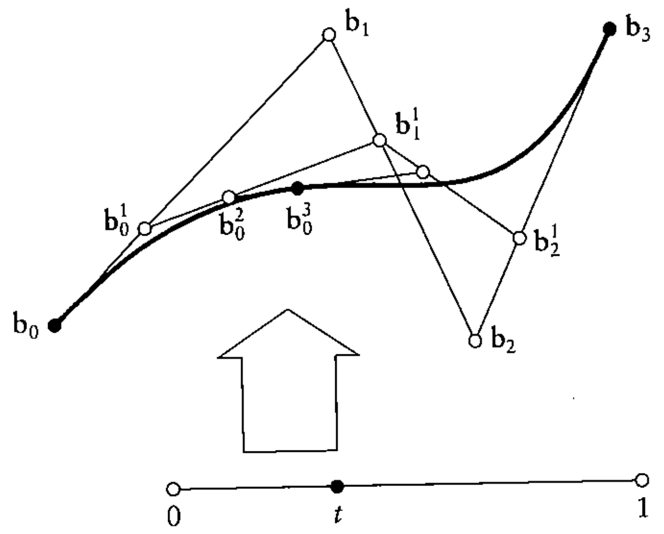

Figure 4.2 The de Casteljau algorithm: the point $\mathbf{b}_0^3(t)$ is obtained from repeated linear interpolation. The cubic case $n = 3$ is shown for $t = 1/3$.

Example 4.1 **Computing a point on a Bézier curve with the de Casteljau algorithm.**

A de Casteljau scheme for a planar cubic and for $t = \frac{1}{2}$:

$$\begin{array}{cccc}
\begin{bmatrix}0\\0\end{bmatrix} & & & \\
\begin{bmatrix}0\\2\end{bmatrix} & \begin{bmatrix}0\\1\end{bmatrix} & & \\
\begin{bmatrix}8\\2\end{bmatrix} & \begin{bmatrix}4\\2\end{bmatrix} & \begin{bmatrix}2\\\frac{3}{2}\end{bmatrix} & \\
\begin{bmatrix}4\\0\end{bmatrix} & \begin{bmatrix}6\\1\end{bmatrix} & \begin{bmatrix}5\\\frac{3}{2}\end{bmatrix} & \begin{bmatrix}\frac{7}{2}\\\frac{3}{2}\end{bmatrix}
\end{array}$$

storage, however: it is sufficient to use the left column only and to overwrite it appropriately.

For a numerical example, see Example 4.1. Figure 4.3 shows 60 evaluations of a Bézier curve. The intermediate points $\mathbf{b}_i^r$ are also plotted and connected.[4]

4 Although the control polygon of the figure is symmetric, the plot is not. This is due to the organization of the plotting algorithm.

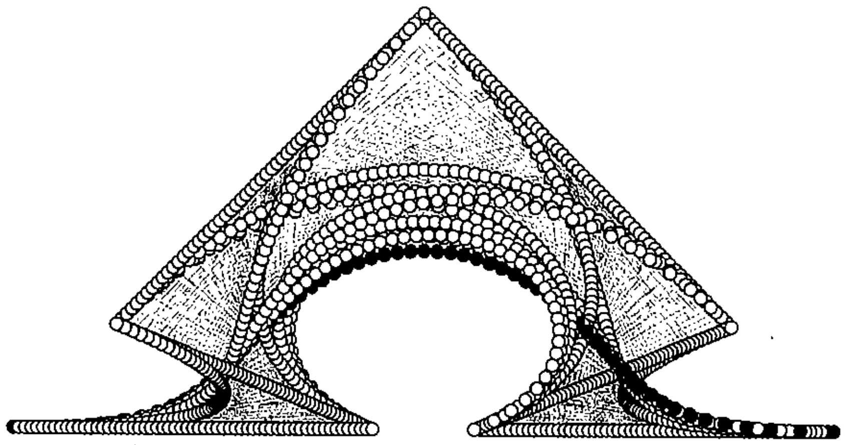

Figure 4.3 The de Casteljau algorithm: 60 points are computed on a degree six curve; all intermedediate points $\mathbf{b}_i^r$ are shown.

4.3 Some Properties of Bézier Curves

The de Casteljau algorithm allows us to infer several important properties of Bézier curves. We will infer these properties from the geometry underlying the algorithm. In the next chapter, we will show how they can also be derived analytically.

Affine invariance. Affine maps were discussed in Section 2.2. They are in the tool kit of every CAD system: objects must be repositioned, scaled, and so on. An important property of Bézier curves is that they are invariant under affine maps, which means that the following two procedures yield the same result: (1) first, compute the point $\mathbf{b}^n(t)$ and then apply an affine map to it; (2) first, apply an affine map to the control polygon and then evaluate the mapped polygon at parameter value t.

Affine invariance is, of course, a direct consequence of the de Casteljau algorithm: the algorithm is composed of a sequence of linear interpolations (or, equivalently, of a sequence of affine maps). These are themselves affinely invariant, and so is a finite sequence of them.

Let us discuss a practical aspect of affine invariance. Suppose we plot a cubic curve $\mathbf{b}^3$ by evaluating at 100 points and then plotting the resulting point array. Suppose now that we would like to plot the curve after a rotation has been applied to it. We can take the 100 computed points, apply the rotation to each of them, and plot. Or, we can apply the rotation to the 4 control points, then evaluate 100 times and plot. The first method needs 100 applications of the rotation, whereas the second needs only 4!

Affine invariance may not seem to be a very exceptional property for a useful curve scheme; in fact, it is not straightforward to think of a curve scheme that does not have it (exercise!). It is perhaps worth noting that Bézier curves do *not* enjoy another, also very important, property: they are not *projectively invariant*. Projective maps are used in computer graphics when an object is to be rendered realistically. So if we try to make life easy and simplify a perspective map of a Bézier curve by mapping the control polygon and then computing the curve, we have actually cheated: that curve is not the perspective image of the original curve! More details on perspective maps can be found in Chapter 12.

Invariance under affine parameter transformations. Very often, one thinks of a Bézier curve as being defined over the interval [0, 1]. This is done because it is convenient, not because it is necessary: the de Casteljau algorithm is "blind" to the actual interval that the curve is defined over because it uses ratios only. One may therefore think of the curve as being defined over any arbitrary interval $a \le u \le b$ of the real line—after the introduction of local coordinates $t = (u - a)/(b - a)$, the algorithm proceeds as usual. This property is inherited from the linear interpolation process (3.9). The corresponding generalized de Casteljau algorithm is of the form:

$$\mathbf{b}_i^r(u) = \frac{b-u}{b-a}\mathbf{b}_i^{r-1}(u) + \frac{u-a}{b-a}\mathbf{b}_{i+1}^{r-1}(u). \tag{4.4}$$

The transition from the interval [0, 1] to the interval $[a, b]$ is an *affine map*. Therefore, we can say that Bézier curves are invariant under affine parameter transformations. Sometimes, one sees the term *linear parameter transformation* in this ~~con~~text, but this terminology is not quite correct: the transformation of the interval [0, 1] to $[a, b]$ typically includes a translation, which is not a linear map.

Convex hull property. For $t \in [0, 1]$, $\mathbf{b}^n(t)$ lies in the convex hull (see Figure 2.3) of the control polygon. This follows since every intermediate $\mathbf{b}_i^r$ is obtained as a convex barycentric combination of previous $\mathbf{b}_j^{r-1}$—at no step of the de Casteljau algorithm do we produce points outside the convex hull of the $\mathbf{b}_i$.

A simple consequence of the convex hull property is that a planar control polygon always generates a planar curve.

The importance of the convex hull property lies in what is known as *interference checking*. Suppose we want to know if two Bézier curves intersect each other—for example, each might represent the path of a robot arm, and our aim is to make sure that the two paths do not intersect, thus avoiding expensive collisions of the robots. Instead of actually computing a possible intersection, we can perform a much cheaper test: circumscribe the smallest possible box around the control polygon of each curve such that it has its edges parallel to some coordinate system. Such boxes are called *minmax boxes*, since their faces are created by the minimal and maximal coordinates of the control polygons. Clearly each box contains its control polygon, and, by the convex hull property, also the corresponding Bézier curve. If we can verify that the two boxes do not overlap (a trivial test), we are assured that the two curves do not intersect. If the boxes do overlap, we would have to perform more checks on the curves. The possibility for a quick decision of no interference is extremely important, since in practice one often has to check one object against thousands of others, most of which can be labeled "no interference" by the minmax box test.[5]

Endpoint interpolation. The Bézier curve passes through $\mathbf{b}_0$ and $\mathbf{b}_n$: we have $\mathbf{b}^n(0) = \mathbf{b}_0, \mathbf{b}^n(1) = \mathbf{b}_n$. This is easily verified by writing down the scheme (4.3) for the cases $t = 0$ and $t = 1$. In a design situation, the endpoints of a curve are certainly two very important points. It is therefore essential to have direct control over them, which is assured by endpoint interpolation.

Designing with Bézier curves. Figure 4.4 shows two Bézier curves. From the inspection of these examples, one gets the impression that in some sense the Bézier curve "mimics" the Bézier polygon—this statement will be made more precise later. It is the reason Bézier curves provide such a handy tool for the *design* of curves: to reproduce the shape of a hand-drawn curve, it is sufficient to specify a control polygon that somehow "exaggerates" the shape of the curve. One lets the computer draw the Bézier curve defined by the polygon, and, if necessary, adjusts the location (possibly also the number) of the polygon vertices. Typically, an experienced person will reproduce a given curve after two to three iterations of this *interactive* procedure.

5 It is possible to create volumes (or areas, in the 2D case) that hug the given curve closer than the minmax box does. See Sederberg et al. [560].

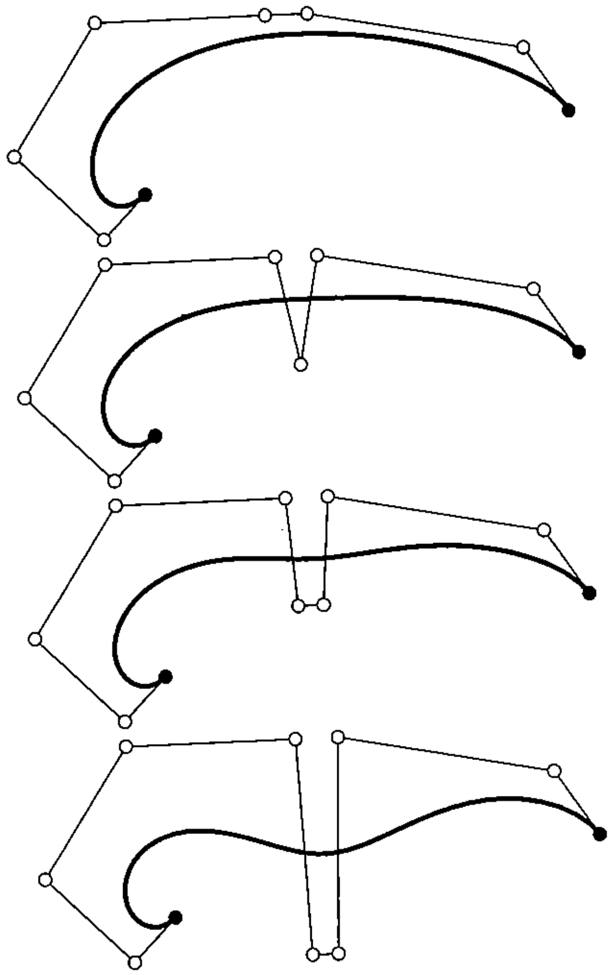

Figure 4.4 Bézier curves: some examples.

4.4 The Blossom

In recent years, a new way to look at Bézier curves has been developed; it is called the principle of *blossoming*. This principle was independently developed by de Casteljau [147] and Ramshaw [498], [499]. Other literature includes Seidel [562], [565], [566]; DeRose and Goldman [165]; Boehm [75]; Lee [379]; and Gallier [252].

Blossoms were introduced in Section 3.4. They are closely related to the de Casteljau algorithm: in column r, do not again perform a de Casteljau step for parameter value t, but use a new value t_r. Restricting ourselves to the cubic case, we obtain:

$$\begin{array}{llll} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0^1[t_1] & & \\ \mathbf{b}_2 & \mathbf{b}_1^1[t_1] & \mathbf{b}_0^2[t_1,t_2] & \\ \mathbf{b}_3 & \mathbf{b}_2^1[t_1] & \mathbf{b}_1^2[t_1,t_2] & \mathbf{b}_0^3[t_1,t_2,t_3]. \end{array} \tag{4.5}$$

The resulting point $\mathbf{b}_0^3[t_1, t_2, t_3]$ is now a function of three independent variables; thus it no longer traces out a curve, but a region of $\mathbb{E}^3$. This trivariate function $\mathbf{b}[\cdot,\cdot,\cdot]$ is called the blossom from Section 3.4. The original curve is recovered if we set all three arguments equal: $t = t_1 = t_2 = t_3$.

To understand the blossom better, we now evaluate it for several special arguments. We already know, of course, that $\mathbf{b}[0,0,0] = \mathbf{b}_0$ and $\mathbf{b}[1,1,1] = \mathbf{b}_3$. Let us start with $[t_1, t_2, t_3] = [0, 0, 1]$. The scheme (4.5) reduces to:

$$\begin{array}{llll} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0 & & \\ \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_0 & \\ \mathbf{b}_3 & \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_1 = \mathbf{b}[0,0,1]. \end{array} \tag{4.6}$$

Similarly, we can show that $\mathbf{b}[0,1,1] = \mathbf{b}_2$. Thus the original Bézier points can be found by evaluating the curve's blossom at arguments consisting only of 0's and 1's.

But the remaining entries in (4.3) may also be written as values of the blossom for special arguments. For instance, setting $[t_1, t_2, t_3] = [0, 0, t]$, we have the scheme

$$\begin{array}{llll} \mathbf{b}_0 & & & \\ \mathbf{b}_1 & \mathbf{b}_0 & & \\ \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_0 & \\ \mathbf{b}_3 & \mathbf{b}_2 & \mathbf{b}_1 & \mathbf{b}_0^1 = \mathbf{b}[0,0,t]. \end{array} \tag{4.7}$$

Continuing in the same manner, we may write the complete scheme (4.3) as:

$$\begin{array}{llll} \mathbf{b}_0 = \mathbf{b}[0,0,0] & & & \\ \mathbf{b}_1 = \mathbf{b}[0,0,1] & \mathbf{b}[0,0,t] & & \\ \mathbf{b}_2 = \mathbf{b}[0,1,1] & \mathbf{b}[0,t,1] & \mathbf{b}[0,t,t] & \\ \mathbf{b}_3 = \mathbf{b}[1,1,1] & \mathbf{b}[t,1,1] & \mathbf{b}[t,t,1] & \mathbf{b}[t,t,t]. \end{array} \tag{4.8}$$

This is easily generalized to arbitrary degrees, where we can also express the Bézier points as blossom values:

$$\mathbf{b}_i = \mathbf{b}[0^{<n-i>}, 1^{<i>}], \tag{4.9}$$

where $t^{<r>}$ means that t appears r times as an argument. For example, $\mathbf{b}[0^{<1>}, t^{<2>}, 1^{<0>}] = \mathbf{b}[0, t, t]$.

The de Casteljau recursion (4.2) can now be expressed in terms of the blossom $\mathbf{b}$:

$$\mathbf{b}[0^{<n-r-i>}, t^{<r>}, 1^{<i>}] = (1-t)\mathbf{b}[0^{<n-r-i+1>}, t^{<r-1>}, 1^{<i>}] + t\mathbf{b}[0^{<n-r-i>}, t^{<r-1>}, 1^{<i+1>}]. \tag{4.10}$$

The point on the curve is given by $\mathbf{b}[t^{<n>}]$.

We may also consider the blossom of a Bézier curve that is not defined over $[0, 1]$ but over the more general interval $[a, b]$. Proceeding exactly as above—but now using (4.4)—we find that the Bézier points $\mathbf{b}_i$ are found as the blossom values

$$\mathbf{b}_i = \mathbf{b}[a^{<n-i>}, b^{<i>}]. \tag{4.11}$$

Thus a cubic over $u \in [a, b]$ has Bézier points $\mathbf{b}[a, a, a], \mathbf{b}[a, a, b], \mathbf{b}[a, b, b], \mathbf{b}[b, b, b]$. If the original Bézier curve was defined over $[0, 1]$, the Bézier points of the one corresponding to $[a, b]$ are simply found by four calls to a blossom routine! See also Figure 4.5.

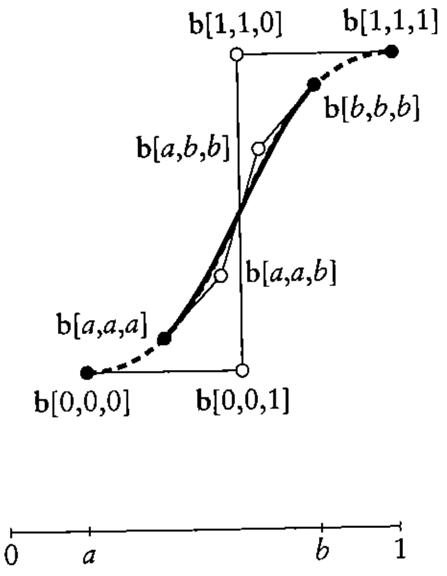

Figure 4.5 Subdivision: the relevant blossom values.

We may also find explicit formulas for blossoms; here is the case of a cubic:

$$\begin{aligned}
&\mathbf{b}[t_1, t_2, t_3] \\
&= (1-t_1)\mathbf{b}[0, t_2, t_3] + t_1\mathbf{b}[1, t_2, t_3] \\
&= (1-t_1)\big[(1-t_2)\mathbf{b}[0, 0, t_3] + t_2\mathbf{b}[0, 1, t_3]\big] + t_1\big[(1-t_2)\mathbf{b}[0, 1, t_3] \\
&\quad + t_2\mathbf{b}[1, 1, t_3]\big] \\
&= \mathbf{b}[0, 0, 0](1-t_1)(1-t_2)(1-t_3) \\
&\quad + \mathbf{b}[0, 0, 1]\big[(1-t_1)(1-t_2)t_3 + (1-t_1)t_2(1-t_3) + t_1(1-t_2)(1-t_3)\big] \\
&\quad + \mathbf{b}[0, 1, 1]\big[t_1t_2(1-t_3) + t_1(1-t_2)t_3 + (1-t_1)t_2t_3\big] \\
&\quad + \mathbf{b}[1, 1, 1]t_1t_2t_3.
\end{aligned}$$

For each step, we have exploited the fact that blossoms are multiaffine, following the inductive proof of the Leibniz equation (3.22).

We should add that every multivariate polynomial function may be interpreted as the blossom of a Bézier curve—as long as it is both symmetric and multiaffine.

4.5 Implementation

The header of the de Casteljau algorithm program is:

```
float decas(degree,coeff,t)
/*   uses  de Casteljau to compute one coordinate
     value of a  Bezier curve. Has to be called
     for each coordinate  (x,y, and/or z) of a control polygon.
Input:   degree:  degree of curve.
         coeff:   array with coefficients of curve.
         t:       parameter value.
Output: coordinate value.
*/
```

This procedure invites several comments. First, we see that it requires the use of an auxiliary array `coeffa`. Moreover, this auxiliary array has to be filled for each function call! So on top of the already high computational cost of the de Casteljau algorithm, we add another burden to the routine, keeping it from being very efficient. A faster evaluation method is given at the end of the next chapter.

To plot a Bézier curve, we would then call the routine several times:

```
void bez_to_points(degree,npoints,coeff,points)
/*     Converts Bezier curve into point sequence. Works on
       one coordinate only.
 Input:   degree:  degree of curve.
          npoints: # of coordinates to be generated. (counting
                   from 0!)
          coeff:   coordinates of control polygon.

Output:  points:   coordinates of points on curve.

     Remark: For a 2D curve, this routine needs to be called twice,
          once for the x-coordinates and once for y.
*/
```

The last subroutine has to be called once for each coordinate, that is, two or three times. The main program `decasmain.c` on the enclosed disk gives an example of how to use it and how to generate postscript output.

4.6 Problems

1 Suppose a planar Bézier curve has a control polygon that is symmetric with respect to the y-axis. Is the curve also symmetric with respect to the y-axis? Be sure to consider the control polygon $(-1,0), (0,1), (1,1), (0,2), (0,1), (-1,1), (0,2), (0,1), (1,0)$. Generalize to other symmetry properties.

2 Use the de Casteljau algorithm to design a curve of degree four that has its middle control point on the curve. More specifically, try to achieve

$$b_2 = b_0^4\left(\frac{1}{2}\right).$$

Five collinear control points are a solution; try to be more ambitious!

* 3 The de Casteljau algorithm may be formulated as

$$B[b_0,\ldots,b_n;t] = (1-t)B[b_0,\ldots,b_{n-1};t] + tB[b_1,\ldots,b_n;t].$$

Show that the computation count is exponential (in terms of the degree) if you implement such a recursive algorithm in a language like C.

*** 4** Show that every nonplanar cubic in $\mathbb{E}^3$ can be obtained as an affine map of the *standard cubic* (see Boehm [70])

$$\mathbf{x}(t) = \begin{bmatrix} t \\ t^2 \\ t^3 \end{bmatrix}.$$

P1 Write an experimental program that replaces $(1-t)$ and t in the recursion (4.2) by $[1-f(t)]$ and $f(t)$, where f is some "interesting" function. Change the routine decas accordingly and comment on your results.

P2 Rewrite the routine decas to handle blossoms. Evaluate and plot for some "interesting" arguments.

P3 Experiment with the data set `outline_2D.dat` on the floppy: try to recapture its shape using one, two, and four Bézier curves. These curves should have decreasing degrees as you use more of them.

P4 Then repeat the previous problem with `outline_3D.dat`. This data set is three dimensional, and you will have to use (at least) two views as you approximate the data points. The points, by the way, are taken from the outline of a high heel shoe sole.

The Bernstein Form of a Bézier Curve

5

Bézier curves can be defined by a recursive algorithm, which is how de Casteljau first developed them. It is also necessary, however, to have an *explicit* representation for them; this will facilitate further theoretical development considerably.

5.1 Bernstein Polynomials

We will express Bézier curves in terms of *Bernstein polynomials*, defined explicitly by

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \tag{5.1}$$

where the binomial coefficients are given by

$$\binom{n}{i} = \begin{cases} \frac{n!}{i!(n-i)!} & \text{if} \quad 0 \le i \le n \\ 0 & \text{else.} \end{cases}$$

There is a fair amount of literature on these polynomials. We cite just a few: Bernstein [53], Lorentz [399], Davis [133], and Korovkin [364]. An extensive bibliography is given in Gonska and Meier [269].

Before we explore the importance of Bernstein polynomials to Bézier curves, let us first examine them more closely. One of their important properties is that they satisfy the following recursion:

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) \tag{5.2}$$

with

$$B_0^0(t) \equiv 1 \tag{5.3}$$

and

$$B_j^n(t) \equiv 0 \quad \text{for} \quad j \notin \{0, \ldots, n\}. \tag{5.4}$$

The proof is simple:

$$\begin{aligned} B_i^n(t) &= \binom{n}{i} t^i (1-t)^{n-i} \\ &= \binom{n-1}{i} t^i (1-t)^{n-i} + \binom{n-1}{i-1} t^i (1-t)^{n-i} \\ &= (1-t) B_i^{n-1}(t) + t B_{i-1}^{n-1}(t). \end{aligned}$$

Another important property is that Bernstein polynomials form a *partition of unity*:

$$\sum_{j=0}^{n} B_j^n(t) \equiv 1. \tag{5.5}$$

This fact is proved with the help of the binomial theorem:

$$1 = [t + (1-t)]^n = \sum_{j=0}^{n} \binom{n}{j} t^j (1-t)^{n-j} = \sum_{j=0}^{n} B_j^n(t).$$

Figure 5.1 shows the family of the four cubic Bernstein polynomials. Note that the B_i^n are nonnegative over the interval [0, 1].

We are now ready to see why Bernstein polynomials are important for the development of Bézier curves. Recall that a Bézier curve may be written as $\mathbf{b}[t^{<n>}]$ in blossom form. Since $t = (1-t) \cdot 0 + t \cdot 1$, the blossom may be expressed as $\mathbf{b}[(1-t) \cdot 0 + t \cdot 1)^{<n>}]$, and now the Leibniz formula (3.22) directly yields

$$\mathbf{b}(t) = \mathbf{b}[t^{<n>}] = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t) \tag{5.6}$$

since $\mathbf{b}_i = \mathbf{b}[0^{<n-i>}, 1^{<i>}]$ according to (4.9).

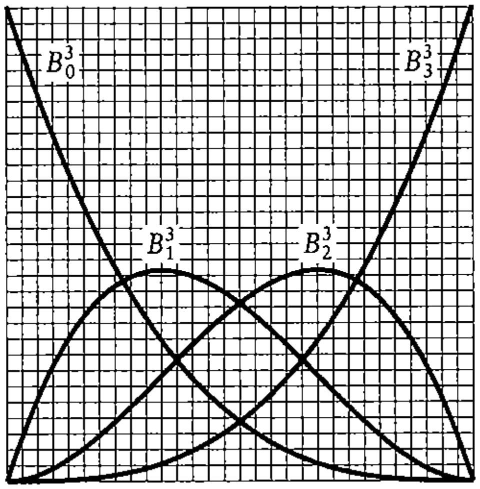

Figure 5.1 Bernstein polynomials: the cubic case.

Similarly, the intermediate de Casteljau points $\mathbf{b}_i^r$ can be expressed in terms of Bernstein polynomials of degree r:

$$\mathbf{b}_i^r(t) = \sum_{j=0}^{r} \mathbf{b}_{i+j} B_j^r(t). \tag{5.7}$$

This follows directly from

$$\mathbf{b}_i^r(t) = \mathbf{b}[0^{<n-r-i>}, t^{<r>}, 1^{<i>}]$$

and the Leibniz formula.

Equation (5.7) shows exactly how the intermediate point $\mathbf{b}_i^r$ depends on the given Bézier points $\mathbf{b}_i$. Figure 5.2 shows how these intermediate points form Bézier curves themselves.

With the intermediate points $\mathbf{b}_i^r$ at hand, we can write a Bézier curve in the form

$$\mathbf{b}^n(t) = \sum_{i=0}^{n-r} \mathbf{b}_i^r(t) B_i^{n-r}(t). \tag{5.8}$$

This is to be interpreted as follows: first, compute r levels of the de Casteljau algorithm with respect to t. Then, interpret the resulting points $\mathbf{b}_i^r(t)$ as control points of a Bézier curve of degree $n - r$ and evaluate it at t.

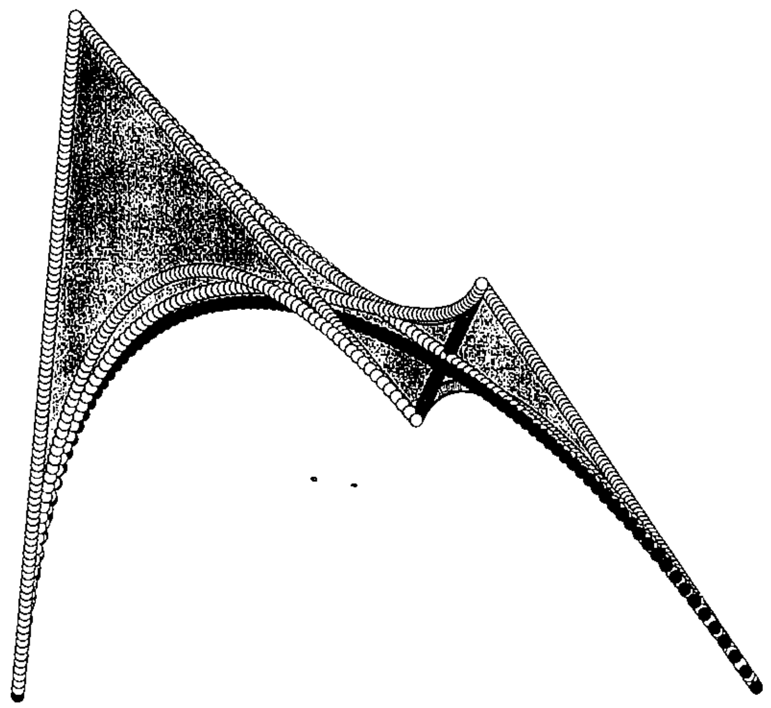

Figure 5.2 The de Casteljau algorithm: 50 points are computed on a quartic curve, and the intermediate points $\mathbf{b}_i^r$ are connected.

5.2 Properties of Bézier Curves

Many of the properties in this section have already appeared in Chapter 4. They were derived using geometric arguments. We shall now rederive several of them, using algebraic arguments. If the same heading is used here as in Chapter 3, the reader should look there for a complete description of the property in question.

Affine invariance. Barycentric combinations are invariant under affine maps. Therefore, (5.5) gives the algebraic verification of this property. We note again that this does not imply invariance under perspective maps!

Invariance under affine parameter transformations. Algebraically, this property reads

$$\sum_{i=0}^{n} \mathbf{b}_i B_i^n(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n\left(\frac{u-a}{b-a}\right). \tag{5.9}$$

Convex hull property. This follows, since for $t \in [0,1]$, the Bernstein polynomials are nonnegative. They sum to one as shown in (5.5). For values of t outside $[0,1]$, the convex hull property does not hold; Figure 5.3 illustrates.

8

B-Spline Curves

B-splines were investigated as early as the nineteenth century by N. Lobachevsky (see Renyi [506], p. 165); they were constructed as convolutions of certain probability distributions.[1] In 1946, I. J. Schoenberg [542] used B-splines for statistical data smoothing, and his paper started the modern theory of spline approximation. For the purposes of this book, the discovery of the recurrence relations for B-splines by C. de Boor [137], M. Cox [129], and L. Mansfield was one of the most important developments in this theory. The recurrence relations were first used by Gordon and Riesenfeld [284] in the context of parametric B-spline curves.

This chapter presents a theory for arbitrary degree B-spline curves. The original development of these curves makes use of d~~ivide~~d differences and is mathematically involved and numerically unstable; see de Boor [138] or Schumaker [546]. A different approach to B-splines was taken by de Boor and Höllig [143]; they used the recurrence relations for B-splines as the starting point for the theory. In this chapter, the theory of B-splines is based on an even more fundamental concept: the *blossoming* method proposed by L. Ramshaw [498] and, in a different form, by P. de Casteljau [147]. More literature on blossoms: Gallier [252], Boehm and Prautzsch [87].

1 However, those were only defined over a very special knot sequence.

8.1 Motivation

B-spline curves consist of many polynomial pieces, offering much more versatility than do Bézier curves. Many B-spline curve properties can be understood by considering just one polynomial piece—that is how we start this chapter.

The Bézier points of a quadratic Bézier curve may be written as blossom values

$$\mathbf{b}[0,0], \mathbf{b}[0,1], \mathbf{b}[1,1].$$

Based on this, we could get the de Casteljau algorithm by repeated use of the identity $t = (1-t)\cdot 0 + t \cdot 1$. The pairs $[0,0], [0,1], [1,1]$ may be viewed as being obtained from the sequence $0, 0, 1, 1$ by taking successive pairs.

Let us now generalize the sequence $0, 0, 1, 1$ to a sequence u_0, u_1, u_2, u_3. The quadratic blossom $\mathbf{b}[u,u]$ may be written as

$$\begin{aligned}\mathbf{b}[u,u] &= \frac{u_2-u}{u_2-u_1}\mathbf{b}[u_1,u] + \frac{u-u_1}{u_2-u_1}\mathbf{b}[u,u_2]\\ &= \frac{u_2-u}{u_2-u_1}\left(\frac{u_2-u}{u_2-u_0}\mathbf{b}[u_0,u_1] + \frac{u-u_0}{u_2-u_0}\mathbf{b}[u_1,u_2]\right)\\ &+ \frac{u-u_1}{u_2-u_1}\left(\frac{u_3-u}{u_3-u_1}\mathbf{b}[u_1,u_2] + \frac{u-u_1}{u_3-u_1}\mathbf{b}[u_2,u_3]\right).\end{aligned}$$

This uses the identity

$$u = \frac{u_2-u}{u_2-u_1}u_1 + \frac{u-u_1}{u_2-u_1}u_2$$

for the first step and the two identities

$$u = \frac{u_2-u}{u_2-u_0}u_0 + \frac{u-u_0}{u_2-u_0}u_2$$

and

$$u = \frac{u_3-u}{u_3-u_1}u_1 + \frac{u-u_1}{u_3-u_1}u_3$$

for the second step. Note that we successively express u in terms of intervals of growing size.

Starting with the $\mathbf{b}[u_i, u_{i+1}]$ as input control points, we may rewrite this as:

$$\begin{array}{lll} \mathbf{b}[u_0, u_1] & & \\ \mathbf{b}[u_1, u_2] & \mathbf{b}[u_1, u] & \\ \mathbf{b}[u_2, u_3] & \mathbf{b}[u, u_2] & \mathbf{b}[u, u]. \end{array}$$

This is a first instance of the de Boor generalization of the de Casteljau algorithm. See Example 8.1 for a detailed computation.

Figure 8.1 illustrates the algorithm, but using the knot sequence $u_0, u_1, u_2, u_3 = 0, 1, 3, 4$ and $u = 2.0$.

Example 8.1 **The de Boor algorithm for $n = 2$.**

Let $u_0, u_1, u_2, u_3 = 0, 2, 4, 6$. Let the control points be given by

$$\mathbf{b}[u_0, u_1] = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mathbf{b}[u_1, u_2] = \begin{bmatrix} 8 \\ 8 \end{bmatrix}, \quad \mathbf{b}[u_2, u_3] = \begin{bmatrix} 8 \\ 0 \end{bmatrix}.$$

Setting $u = 3$, we now compute the point $\mathbf{b}[3, 3]$. At the first level, we compute two points

$$\mathbf{b}[2, 3] = \frac{4-3}{4-0}\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{3-0}{4-0}\begin{bmatrix} 8 \\ 8 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

and

$$\mathbf{b}[3, 4] = \frac{6-3}{6-2}\begin{bmatrix} 8 \\ 8 \end{bmatrix} + \frac{3-2}{6-2}\begin{bmatrix} 8 \\ 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \end{bmatrix}.$$

Finally,

$$\mathbf{b}[3, 3] = \frac{4-3}{4-2}\begin{bmatrix} 6 \\ 6 \end{bmatrix} + \frac{3-2}{4-2}\begin{bmatrix} 8 \\ 6 \end{bmatrix} = \begin{bmatrix} 7 \\ 6 \end{bmatrix}.$$

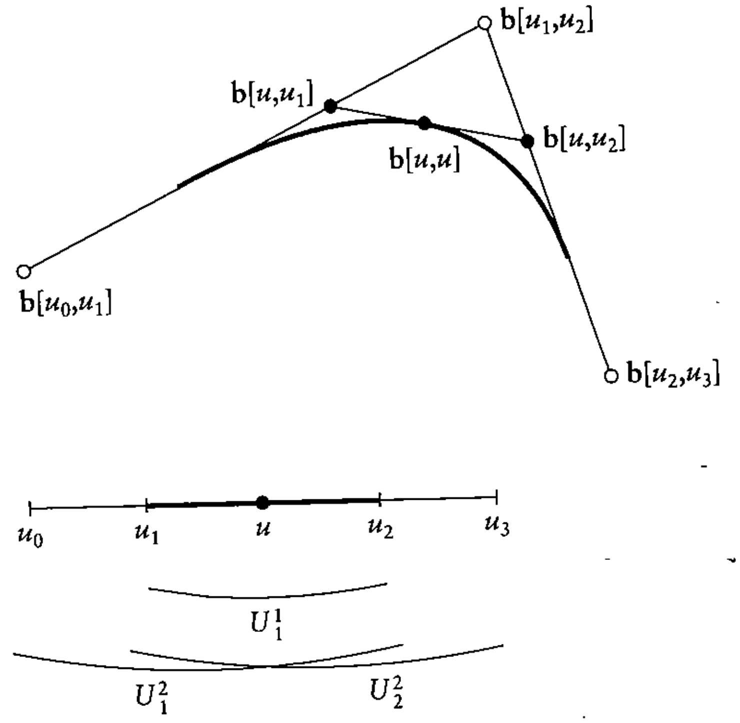

Figure 8.1 The de Boor algorithm: the quadratic case.

8.2 B-Spline Segments

B-spline curves consist of a sequence of polynomial curve segments. In this section, we focus on just one of them.

Let U be an interval $[u_I, u_{I+1}]$ in a sequence $\{u_i\}$ of knots. We define ordered sets U_i^r of successive knots, each containing u_I or u_{I+1}. The set U_i^r is defined such that:

U_i^r consists of $r+1$ successive knots.

u_I is the $(r-i)$th element of U_i^r, with $i=0$ denoting the first of U_i^r's elements.

We also observe

$$U_i^r = U_i^{r+1} \cap U_{i+1}^{r+1}.$$

When the context is unambiguous, we also refer to the U_i^r as *intervals*, having the first and last elements of U_i^r as endpoints. In that context, $U_1^1 = U$. We also define $U = [U_0^0, U_1^0]$ and use the term *interval* only if $U_0^0 \neq U_1^0$.

A degree n curve segment corresponding to the interval U is given by $n+1$ control points $\mathbf{d}_i$ which are defined by

$$\mathbf{d}_i = \mathbf{b}[U_i^{n-1}]; \quad i = 0, \ldots, n. \tag{8.1}$$

The point $\mathbf{x}(u) = \mathbf{b}[u^{<n>}]$ on the curve is recursively computed as

$$\mathbf{d}_i^r(u) = \mathbf{b}[u^{<r>}, U_i^{n-1-r}]; \quad r = 1, \ldots, n; i = 0, \ldots, n-r \tag{8.2}$$

with $\mathbf{x}(u) = \mathbf{d}_0^n(u)$.[2] This is known as the *de Boor algorithm* after Carl de Boor see [137]. See Example 8.2 for the case $n = 3$ and Figure 8.2 for an illustration.

Equation (8.2) may alternatively be written as

$$\mathbf{d}_i^r(u) = (1 - t_{i+1}^{n-r+1})\mathbf{d}_i^{r-1} + t_{i+1}^{n-r+1}\mathbf{d}_{i+1}^{r-1}; \quad r = 1, \ldots, n; i = 0, \ldots, n-r, \tag{8.3}$$

where t_{i+1}^{n-r+1} is the local parameter in the interval U_{i+1}^{n-r+1}.

A geometric interpretation is as follows. Each intermediate control polygon leg $\mathbf{d}_i^r, \mathbf{d}_{i+1}^r$ may be viewed as an affine image of U_{i+1}^{n-r+1}. The point $\mathbf{d}_i^{r+1}$ is then the image of u under that affine map.

For the special knot sequence $0^{<n>}, 1^{<n>}$ and $U = [0, 1]$, the de Boor algorithm becomes

$$\mathbf{d}_i^r(u) = \mathbf{b}[u^{<r>}, 0^{<n-r-i>}, 1^{<i>}]; \quad r = 1, \ldots, n; \quad i = 0, \ldots, n-r, \tag{8.4}$$

which is simply the de Casteljau algorithm.

If the parameter u happens to be one of the knots, the algorithm proceeds as before, except that we do not need as many levels of the algorithm. For example, if a quadratic curve segment is defined by $\mathbf{b}[u_0, u_1], \mathbf{b}[u_1, u_2], \mathbf{b}[u_2, u_3]$ and we want to evaluate at $u = u_2$, then two of the intermediate points in the de Boor algorithm are already known, namely, $\mathbf{b}[u_1, u_2]$ and $\mathbf{b}[u_2, u_3]$. From these two, we immediately calculate the desired point $\mathbf{b}[u_2, u_2]$, thus the de Boor algorithm now needs only one level instead of two.

Derivatives of a B-spline curve segment are computed in analogy to the Bézier curve case (5.17)

$$\dot{\mathbf{x}}(u) = n\mathbf{b}[u^{<n-1>}, \vec{1}]. \tag{8.5}$$

Expanding this expression and using the control point notation, this becomes

$$\dot{\mathbf{x}}(u) = \frac{n}{|U|}(\mathbf{d}_1^{n-1} - \mathbf{d}_0^{n-1}), \tag{8.6}$$

2 This notation is different from the one used in previous editions of this book.

Example 8.2 **The de Boor algorithm for $n = 3$.**

Let part of a knot sequence be given by

$$\dots u_3, u_4, u_5, u_6, u_7, u_8, \dots$$

and let $U = [u_5, u_6]$. The standard blossom computation of $\mathbf{b}[u, u, u]$ proceeds as follows:

$$\begin{array}{llll}
\mathbf{b}[u_3, u_4, u_5] & & & \\
\mathbf{b}[u_4, u_5, u_6] & \mathbf{b}[u, u_4, u_5] & & \\
\mathbf{b}[u_5, u_6, u_7] & \mathbf{b}[u, u_5, u_6] & \mathbf{b}[u, u, u_5] & \\
\mathbf{b}[u_6, u_7, u_8] & \mathbf{b}[u, u_6, u_7] & \mathbf{b}[u, u, u_6] & \mathbf{b}[u, u, u].
\end{array}$$

We now write this as

$$\begin{array}{llll}
\mathbf{b}[U_0^2] & & & \\
\mathbf{b}[U_1^2] & \mathbf{b}[u, U_0^1] & & \\
\mathbf{b}[U_2^2] & \mathbf{b}[u, U_1^1] & \mathbf{b}[u, u, U_0^0] & \\
\mathbf{b}[U_3^2] & \mathbf{b}[u, U_2^1] & \mathbf{b}[u, u, U_1^0] & \mathbf{b}[u, u, u].
\end{array}$$

In terms of control points:

$$\begin{array}{llll}
\mathbf{d}_0 & & & \\
\mathbf{d}_1 & \mathbf{d}_0^1 & & \\
\mathbf{d}_2 & \mathbf{d}_1^1 & \mathbf{d}_0^2 & \\
\mathbf{d}_3 & \mathbf{d}_2^1 & \mathbf{d}_1^2 & \mathbf{d}_0^3.
\end{array}$$

The labeling in the first scheme depends on the subscripts of the knots, whereas the last two employ a relative numbering.

where $|U| = U_1^0 - U_0^0$ denotes the length of the interval U. Thus the last two intermediate points $\mathbf{d}_0^{n-1}$ and $\mathbf{d}_1^{n-1}$ span the curve's tangent, in complete analogy to the de Casteljau algorithm.

Higher derivatives follow the same pattern:

$$\frac{\mathrm{d}^r}{\mathrm{d}u^r}\mathbf{x}(u) = \frac{n!}{(n-r)!}\mathbf{b}[u^{<n-r>}, \vec{1}^{<r>}]. \tag{8.7}$$

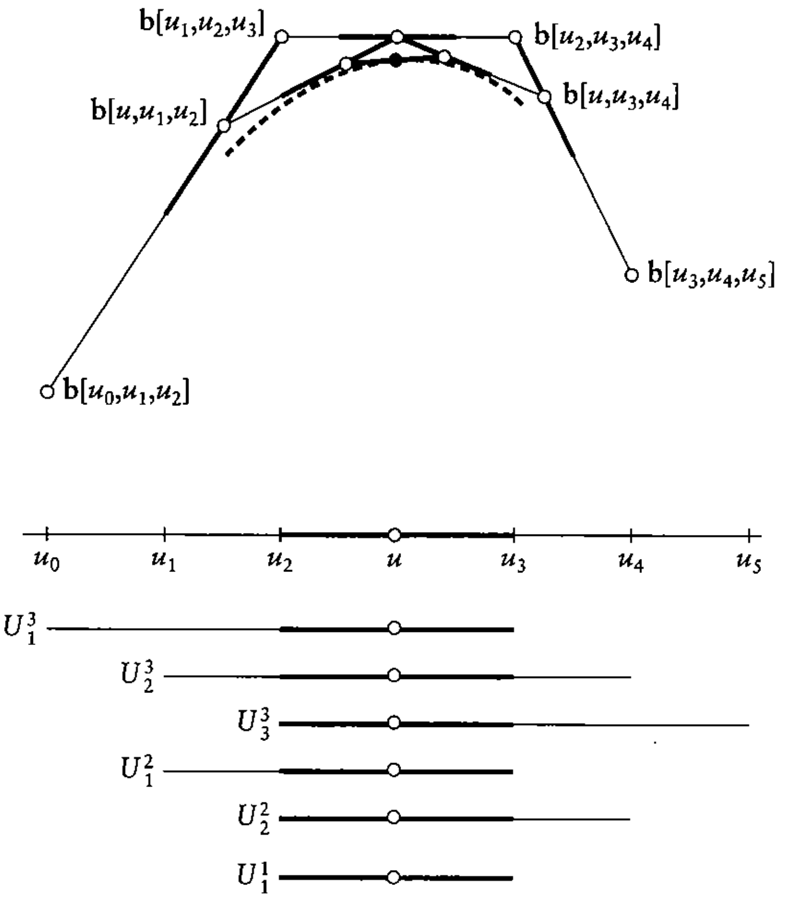

Figure 8.2 The de Boor algorithm: a cubic example. The solid point is the result $\mathbf{b}[u, u, u]$; it is on the line through $\mathbf{b}[u, u, u_2]$ and $\mathbf{b}[u, u, u_3]$.

In the case of Bézier curves, we could use the de Casteljau algorithm for curve evaluation, but we could also write a Bézier curve explicitly using Bernstein polynomials. Since we changed the domain geometry, we will now obtain a different explicit representation, using polynomials[3] P_i^n:

$$\mathbf{x}(u) = \sum_{i=0}^{n} \mathbf{d}_i P_i^n(u). \tag{8.8}$$

The polynomials P_i^n satisfy a recursion similar to the one for Bernstein polynomials, and the following derivation is very similar to that case:

3 These will later become building blocks of B-splines.

$$
\begin{aligned}
\mathbf{x}(u) &= \sum_{i=0}^{n-1} \mathbf{d}_i^1 P_i^{n-1}(u) \\
&= \sum_{i=0}^{n-1}(1 - t_{i+1}^n)\mathbf{d}_i P_i^{n-1}(u) + \sum_{i=0}^{n-1} t_{i+1}^n \mathbf{d}_{i+1} P_i^{n-1}(u) \qquad (8.9)\\
&= \sum_{i=0}^{n}(1 - t_{i+1}^n)\mathbf{d}_i P_i^{n-1}(u) + \sum_{i=1}^{n} t_i^n \mathbf{d}_i P_{i-1}^{n-1}(u)
\end{aligned}
$$

For the first step, we used the de Boor algorithm, letting t_i^n be the local parameter in the interval U_{i+1}^n. For the second step, we used the convention $P_n^{n-1} \equiv 0$ to modify the first term. Using a similar argument: $P_{-1}^{n-1} \equiv 0$, we may extend the second term to start with $i = 0$. We conclude

$$P_i^n(u) = (1 - t_{i+1}^n)P_i^{n-1}(u) + t_i^n P_{i-1}^{n-1}(u). \qquad (8.10)$$

This recursion has to be anchored in order to be useful. This is straightforward for the case $n = 1$:

$$P_0^1(u) = \frac{u_r - U_1^0}{|U|}, \quad P_1^1(u) = \frac{u - U_0^0}{|U|},$$

where $|U| = U_1^0 - U_0^0$. For the special knot sequence $0^{<n>}, 1^{<n>}$ and $U = [0, 1]$, this is the Bernstein recursion.

8.3 B-Spline Curves

A B-spline curve consists of several polynomial curve segments, each of which may be evaluated using a de Boor algorithm. A B-spline curve is defined by

the degree n of each curve segment,

the knot sequence $u_0, \ldots, u_K$, consisting of $K + 1$ knots $u_i \leq u_{i+1}$,

the control polygon $\mathbf{d}_0, \ldots, \mathbf{d}_L$ with $L = K - n + 1$.

Some comments: the numbering of the control points $\mathbf{d}_i$ in this definition is *global*, whereas in Section 8.2 it was *local* relative to an interval U. Each $\mathbf{d}_i$ may be written as a blossom value with n subsequent knots as arguments. Hence the number $L + 1$ of control points equals the number of n-tuples in the knot sequence.

Example 8.3 **Some examples of B-spline curve definitions.**

Let $n = 1$, and let the knot sequence be $0, 1, 2$, hence $K = 2$. There will be control points $\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_3$. The curve's domain is $[u_0, u_3]$, and there are two linear curve segments.
Let $n = 2$ with the knot sequence $0, 0.2, 0.4, 0.5, 0.7, 1$, hence $K = 5$. There will be control points $\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4$ and three quadratic curve segments. If we now change the knot sequence to $0, 0.2, 0.45, 0.45, 0.7, 1$, then the number of curve segments will drop to two.

Each knot may be repeated in the knot sequence up to n times. In some cases it is approriate to simply list those knots multiple times. For other applications, it is better to list the knot only once and record its *multiplicity* in an integer array. For example, the knot sequence 0.0, 0.0, 1.0, 2.0, 3.0, 3.0, 4.0, 4.0 could be stored as 0.0, 1.0, 2.0, 3.0, 4.0 and a multiplicity array 2, 1, 1, 2, 2.

There is a different de Boor algorithm for each curve segment. Each is "started" with a set of U_i^n, that is, by sequences of $n + 1$ knots. In order for local coordinates to be defined in (8.3), no successive $n + 1$ knots may coincide.

In Section 8.2, we assumed that we could find the requisite U_i^n for each interval U. This is possible only if U is "in the middle" of the knot sequence; more precisely, the first possible de Boor algorithm is defined for $U = [u_{n-1}, u_n]$ and the last one is defined for $U = [u_{K-n}, u_{K-n+1}] = [u_{K-n}, u_L]$. We thus call $[u_{n-1}, u_L]$ the *domain* of the B-spline curve. A B-spline curve has as many curve segments as there are nonzero intervals U in the domain. Example 8.3 illustrates these comments.

For more examples of B-spline curves, see Figure 8.3.

Since a B-spline curve consists of a number of polynomial segments, one might ask for the Bézier form of these segments. For a segment $U = [u_I, u_{I+1}]$ of the curve, we simply evaluate its blossom $\mathbf{b}^U$ and obtain the Bézier points $\mathbf{b}_0^U, \ldots, \mathbf{b}_n^U$ as

$$\mathbf{b}_k^U = \mathbf{b}^U[u_I^{<n-k>}, u_{I+1}^{<k>}].$$

An example is shown in Figure 8.4. Several constituent curve pieces of the same curve are shown in Figure 8.5.

When dealing with B-spline curves, it is convenient to treat it as one curve, not just as a collection of polynomial segments. A point on such a curve is denoted by $\mathbf{d}(u)$, with $u \in [u_{n-1}, u_{K-n+1}]$. In order to evaluate, we perform the following steps:

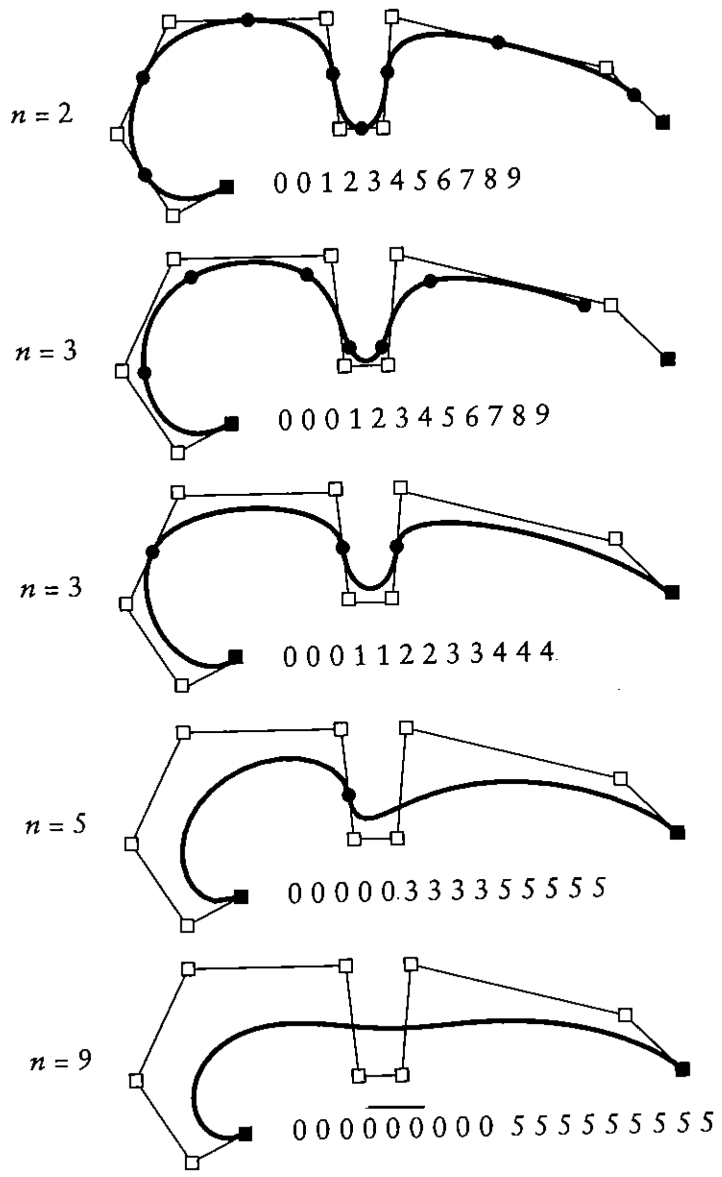

Figure 8.3 B-spline curves: several examples.

1. Find the interval $U = [u_I, u_{I+1})$ that contains u.
2. Find the $n + 1$ control points that are relevant for the interval U. They are, using the global numbering, given by $\mathbf{d}_{I-n+1}, \ldots, \mathbf{d}_{I+1}$.
3. Renumber them as $\mathbf{d}_0, \ldots, \mathbf{d}_n$ and evaluate using the de Boor algorithm (8.3).

In terms of the global numbering of knots, we observe that the intervals U_i^k from the previous section are given by

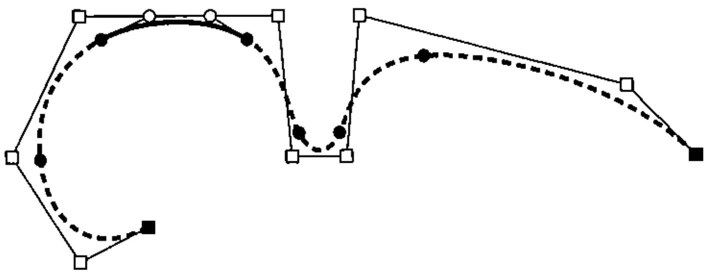

Figure 8.4 Conversion to Bézier form: the Bézier points of a segment of a cubic B-spline curve.

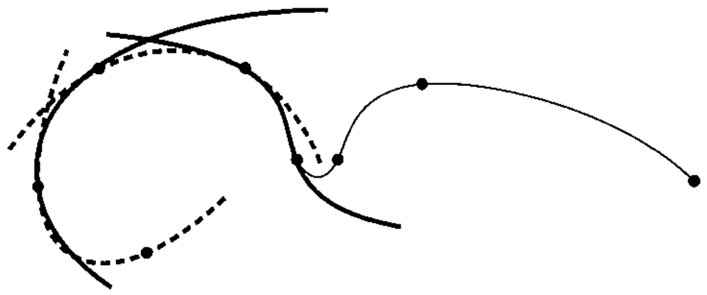

Figure 8.5 Individual curve segments: the first four cubic segments of a cubic B-spline curve are shown, alternating between dashed and black.

$$U_i^k = [u_{I-k+i}, u_{I+i}].$$

The steps in the de Boor algorithm then become

$$\mathbf{d}_i^k(u) = (1-\alpha_i^k)\mathbf{d}_i^{k-1}(u) + \alpha_i^k \mathbf{d}_{i+1}^{k-1}(u)$$

with

$$\alpha_i^k = \frac{u - u_{I-k+i}}{u_{I+i} - u_{I\_k+i}}$$

for $k = r+1, \ldots, n$, and $i = 0, \ldots, n-k$. Here, r denotes the multiplicity of u. (Normally, u is not already in the knot sequence; then, $r = 0$.)

The fact that each curve segment is only affected by $n+1$ control points is called the *local control property*.

We also use the notion of a B-spline blossom $\mathbf{d}[v_1, \ldots, v_n]$, keeping in mind that each domain interval U has its own blossom $\mathbf{b}^U$ and that consequently $\mathbf{d}[v_1, \ldots, v_n]$ is piecewise defined.

B-spline curves enjoy all properties of Bézier curves, such as affine invariance, variation diminution, etc. Some of these properties are more pronounced now because of the local control property. Take the example of the convex hull

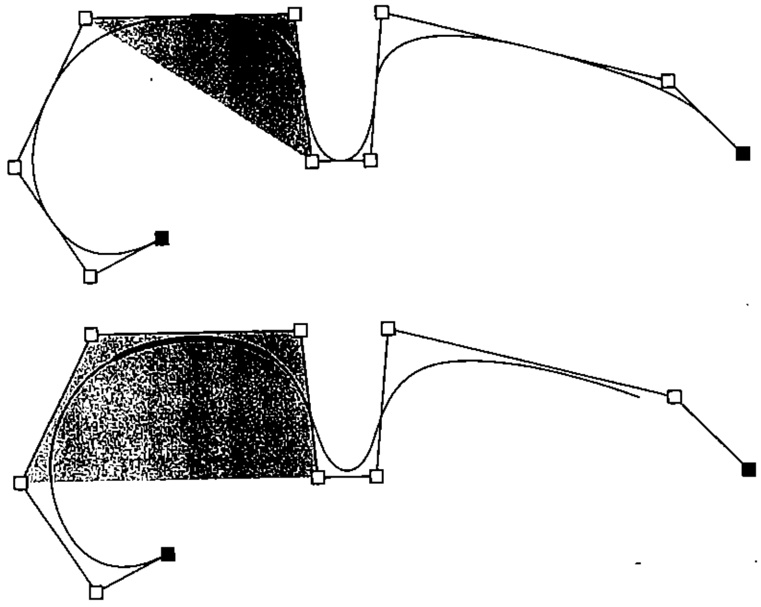
Figure 8.6 The local convex hull property: top, quadratic; bottom, cubic.

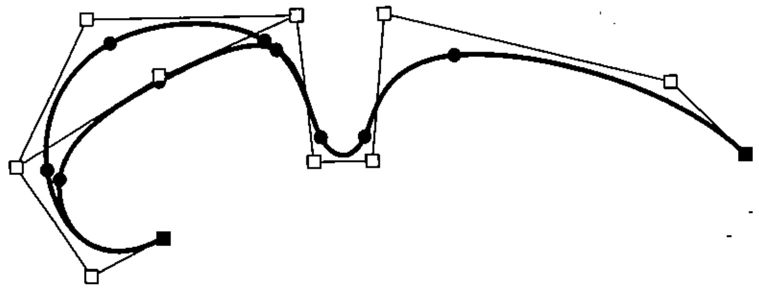
Figure 8.7 The local control property: changing one control point of a cubic B-spline curve has only a local effect.

property: the curve is in the convex hull of its control polygon, but also each segment is in the convex hull of its defining $n+1$ control points; see Figure 8.6.

A consequence of the local control property is that changing one control point will only affect the curve locally. This is illustrated in Figure 8.7.

8.4 **Knot Insertion**

Consider a B-spline curve segment defined over an interval U. It is defined by all blossom values $\mathrm{d}[U_i^{n-1}]$; $i = 0, \ldots, n$ where each n-tuple of successive knots U_i^{n-1} contains at least one of the endpoints of U. If we now split U into two segments by inserting a new knot $\hat{u}$, the curve will have two corresponding