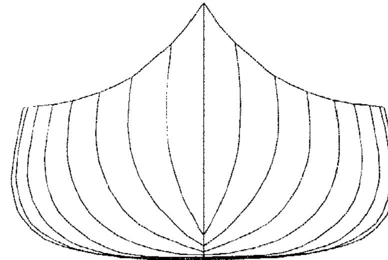
7



Splines and Other Curves for Data Fitting

The word *spline* originally referred to the flexible wood or metal strips used by draftspersons for drawing curves. These physical splines may be held in place with tacks or with lead weights called *ducks* that are used with large splines in boat and ship design. In the past, ship splines were laid out large loft rooms, and this drafting was called *lofting*.

The drawing at the top of this page shows vertical cross-section curves (ribs) for a canoe, along with a vertical line for the keel and two sheer lines. In chapter 12 these vertical cross-section lines will be connected to produce a surface representation of the canoe in three dimensions. Many of the curve types discussed in this chapter can be used to construct "patches" to represent surfaces in space, as developed in chapter 12.

In this chapter we consider a half-dozen of the common types of curves used for design purposes. These and other curves are analyzed more thoroughly in specialized books; see, for example, Bartels et al. (1987), Rogers and Adams (1990), and Farin (1990). One type of curve discussed below, the Bézier curve, was developed for the design of Renault automobiles.

Two general types of curves will be considered. The first type represents an *interpolation* between data points; the curve passes through each point. The second curve type

is an *approximation* curve; it passes through few, if any, of the data points. In this latter case, the data points act as control points to shape the curve, rather than as points on the curve.

This chapter starts with a brief discussion of using high-order polynomials to fit data. This procedure, often described in calculus textbooks, is not useful for design purposes because the curve often has large oscillations between data points. Other interpolation schemes—which are useful for design work—are a circular-arc fit, parabolic blending, and the natural cubic spline. The natural cubic spline is the mathematical representation of the physical spline described above. Also in this chapter, various types of approximation curves are analyzed: uniform quadratic and cubic B-splines, nonuniform (nonrational and rational) B-splines of general order, general-order Bézier curves, and composite cubic Bézier curves.

Although high-order polynomials are discussed in the first section and in the general B-spline or Bézier cases, this chapter emphasizes curves that are constructed with segments described by second- and third-degree polynomials (quadratic and cubic equations), with the segments being connected at *joints* (also called *knots*). Note that the simplest curve fit is the linear case, considered in chapter 3 as a polyline that joins data points with straight-line segments.

The joints that connect curved-line segments may be classified according to the continuity of the derivatives at the joints. If the nth derivative is continuous at the joints, then the overall curve is said to be C^n continuous. The simple polyline is C^0 continuous because values match at the joints, but first derivatives do not. The circular-arc fit and parabolic blend are C^1 continuous, whereas the natural cubic spline is C^2 continuous. For the approximation curves, the quadratic B-spline and the composite cubic Bézier curve considered in this chapter have C^1 continuity, and the cubic B-spline is C^2 continuous.

7.1 HIGH-ORDER POLYNOMIALS

Consider n points at $y_i = f(x_i)$, where the x_i are monotonically increasing x-values. The following Lagrange interpolating polynomial of order $n-1$ passes through each of the n points.

$$y = \sum_{i=0}^{n-1} L_i(x) f(x_i) \tag{7.1}$$

where

$$L_i(x) = \prod_{j=0, j\neq i}^{n-1} \frac{x-x_j}{x_i-x_j}$$

Note that at a vertex point, say $x = x_k$, the polynomials $L_i(x)$ will equal zero for all vertices x_i except for $x_i = x_k$, where $L_k(x) = 1$. Therefore equation 7.1 reduces to $y = f(x_i)$ at the vertex points x_i.

The formula (7.1) is readily implemented by the following subroutine. The coordinates are in the units of screen pixels, and x is increased in steps of five pixels.

```
void highPolyFit(int n,double x[],double y[])
{
    int i,j;
    double xp,yp,L;

    MoveTo((int) x[0],(int) y[0]);
    for (xp=x[0]; xp<x[n-1]+.1; xp=xp+5.)
    {
        yp = 0.;
        for (i=0; i<n; i++)
        {
            L = 1.;
            for (j=0; j<n; j++)
            {
                if (j!=i) L = L*(xp-x[j])/(x[i]-x[j]);
            }
            yp = yp + L*y[i];
        }
        LineTo((int) xp,(int) yp);
    }
}
```

In the above listing, n points are passed to the subroutine by means of the arrays $x[]$ and $y[]$ for the horizontal and vertical coordinates. These arrays have elements $x[0]$ through $x[n-1]$. A horizontal point xp is incremented by a small amount (here five pixels), and the vertical coordinate yp is calculated from equation 7.1.

Although high-order polynomials pass through the vertex points, they usually oscillate too much between the vertex points. This result is not surprising because, for example, x to a large power n will increase rapidly when x is greater than unity. Figure 7.1 shows four different interpolation curves that pass through six vertex points representing data in which the first three points lie on a horizontal line and the last three points are on a higher (larger y-value) horizontal line. For this case the high-order (Lagrange) polynomial is a fifth-order polynomial.

The high-order polynomial shown as curve a in figure 7.1 oscillates widely between vertex points. Even higher oscillations can occur for other vertex configurations. Obviously, high-order polynomials are not suitable for approximating smooth curves in design applications. The other three curves in figure 7.1 interpolate the data in better ways.

A circular-arc fit (curve b) reduces to straight lines along the three colinear points at the beginning of the curve, and also at the end of the curve. It produces an s-shaped curve between the third and fourth vertex points.

Curve c in figure 7.1 represents a parabolic blend, which produces (between the third and fourth vertices) an s-shaped curve with much less deviation from the straight-line slope

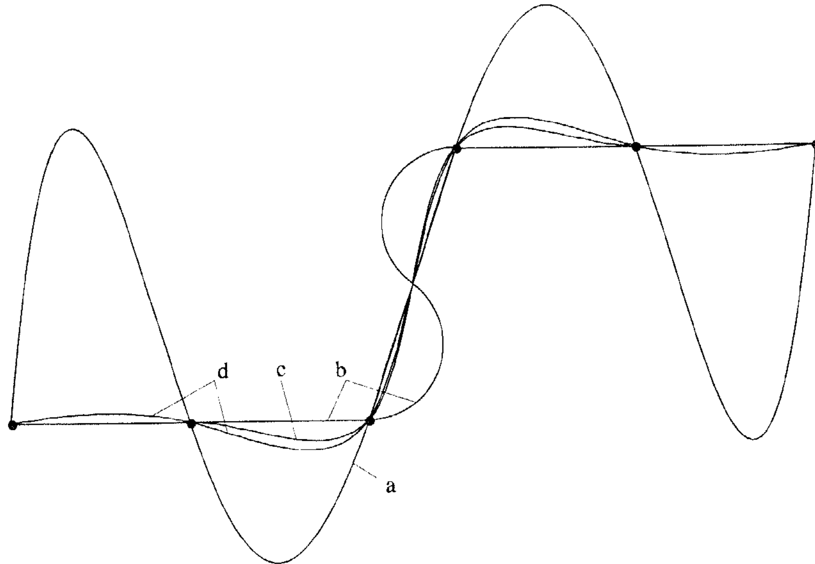


Figure 7.1 Interpolation curves drawn for six vertex points (dots), with y plotted in the vertical and x in the horizontal direction. Curves are shown for (a) a high-order polynomial fit, (b) a circular-arc fit, (c) a parabolic blend, and (d) a natural cubic spline.

than the circular-arc curve. Although this gentler s-shape may be more pleasing, it produces small oscillations in the curve between the other vertices (the deviation from the straight lines between the first two vertices and the last two is too small to be evident in the figure).

The natural cubic spline is represented by curve d in figure 7.1. It oscillates more between the first three and last three vertices than does the parabolic blend, but it has a less pronounced s-shape between the third and fourth vertices. Furthermore, the natural cubic spline has C^2 continuity (continuity of curvature) at the vertices, rather than only C^1 continuity like the circular-arc and parabolic-blend curves.

Figure 7.2 presents four approximation curves along with a polyline that connects the same six vertices as shown in figure 7.1. These approximation curves satisfy convex hull properties; for example, they all lie within the polygon that would be formed if a rubber band were stretched around all of the vertex points in figure 7.2. Accordingly, there are no oscillations in the curves between vertex points.

For approximation curves, the vertex points act as control points that attract the curve even though the curve does not normally pass through the vertex points. The general Bézier curve d is controlled at each point by all of the vertex points, whereas the other curves shown are controlled by the nearest three or four vertex points on any part of the curve. Therefore, for the general Bézier curve, a vertex point does not exhibit much local control over curve shape compared with the other three approximation curves. In figure 7.2 this lack of local control allows the curve to rise much more gradually between the lower and upper vertex points.

All of the curves in figures 7.1 and 7.2 are analyzed in this chapter.

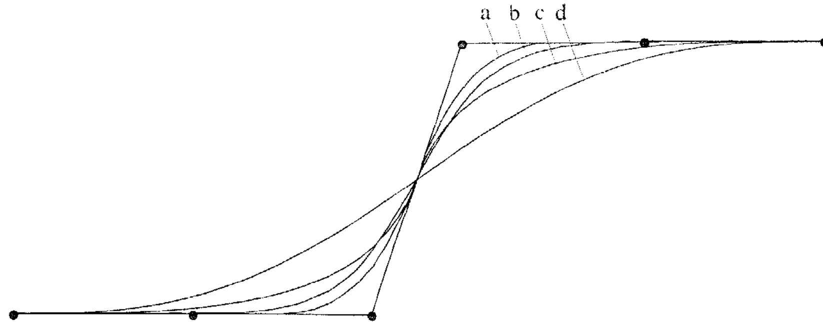


Figure 7.2 Approximation curves drawn for the same six vertex points shown in figure 7.1, with y plotted in the vertical and x in the horizontal direction. Curve a is the uniform quadratic B-spline; b the uniform cubic B-spline; c the composite cubic Bézier curve; and d the general (here fifth-order) Bézier curve.

7.2 CIRCULAR-ARC CURVE FIT

The interpolation curve fit used by AutoCAD consists of a pair of arcs drawn between each pair of vertex points. Because two arcs are used between vertices, this procedure is often called the *biarc method.* The biarc method was developed by Renner and Pochop (1981) for use in the design of BMW automobiles.

Use of the Circular-Arc Fit in AutoCAD

Any polyline drawn in AutoCAD (with the PLINE command) can be changed into a circular-arc fit of the data (vertex) points by using the FIT option under PEDIT. As discussed in chapter 5, polylines are edited as follows.

```
Command: PEDIT
Select polyline: (select with mouse)
Close/Join/Width/Edit vertex/Fit curve/
            Spline curve/Decurve/Undo/eXit <X>: F
Close/Join/Width/Edit vertex/Fit curve/
            Spline curve/Decurve/Undo/eXit <X>: (RETURN)
```

The response of "F" for fit immediately turns the polyline into a circular-arc curve fit. The curve may be changed back to a polyline by entering "D" for decurve.

After a polyline is drawn, the vertex points may be changed with the Edit vertex command under PEDIT.

```
Close/Join/Width/Edit vertex/Fit curve/
            Spline curve/Decurve/Undo/eXit <X>: E
Next/Previous/Break/Insert/Move/Regen/
             Straighten/Tangent/Width/eXit <N>: N
```

When "E" is entered a × symbol appears at the first vertex. This × can be moved to the next vertex by entering "N." After progressing along the vertex points, the polyline can be retraced in the opposite direction by inserting "P" for previous vertex. After selection, a vertex may be moved by entering "M," after which you will be asked for the coordinates of the new position. The edit vertex mode also permits new vertex points to be inserted when "I" is entered. The new vertex will appear between the selected vertex and the next vertex at coordinates specified under the insert option.

Vertex points may be eliminated by using the "straighten" option in the edit vertex mode. For example, the third and fourth vertex points on a polyline may be eliminated by entering "S" for straighten when the × symbol is on the second vertex. Then the × is moved to the fifth vertex, where "G" for go is entered. This action eliminates the third and fourth vertex points, and draws a straight polyline segment between the second and fifth vertex points (the fifth vertex is now the third vertex in the new polyline).

Manufacturing constraints often restrict mechanical parts and other objects to designs that have drawings consisting of straight lines and circular arcs. The crank arm drawn in figure 7.3 illustrates that the circular-arc fit is efficient for drawing these items (this is the same crank arm shown in figure 4.3). Five polylines are used to draw the entire crank arm: one for the entire outside edge, one for the slot that has semicircular ends, and three for the three circles.

Only eight points define the slot with semicircular ends. Because the circular-arc fit draws a straight line between any three colinear points, only three points are required for a straight line segment. Also, a circle is drawn through a closed polyline consisting of four vertex points lying anywhere on the circle (note that one of the three circles in the figure does not have all of its vertex points placed at opposite ends of diameters).

Because of the 75° angle in the crank arm, construction lines and circles were used to locate many of the polyline points in figure 7.3; however, the number of resulting polyline vertex points needed to complete the drawing is remarkably small. Other parts would be simpler to draw. For example, if the crank arm had a 90° angle, then the polylines could have been drawn directly on a rectangular grid.

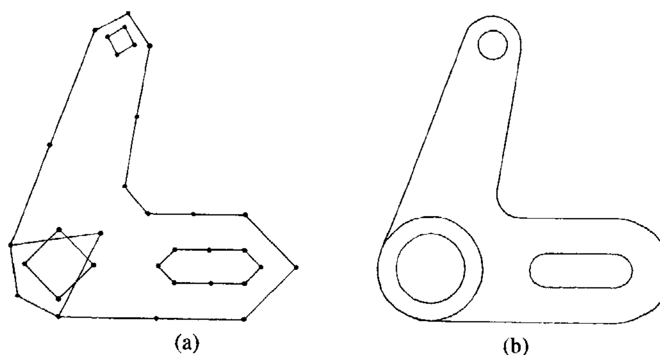


Figure 7.3 A crank arm drawn on AutoCAD with five polylines: (a) the polylines and vertex points and (b) the polylines after the circular-arc curve fit.

The warehouse ceiling structure shown in figure 7.4 was drawn with the circular-arc fit command in AutoCAD. The graceful concrete ribs are arrayed around the supporting columns, shown as filled squares in this plan view. First, one octant is drawn (figure 7.4a); then it is mirrored across a 45° line; then the resulting quadrant is put into a polar array around the column center; and finally all of the curves around one column are put into a rectangular array to produce the desired number of columns in the final illustration. (Of course, the BLOCK command could be used in place of the ARRAY command to save memory.) Proper matching along the 45° mirror line is achieved by using the Tangent option while in the Edit vertex mode of the PEDIT command. The tangents of the first and last

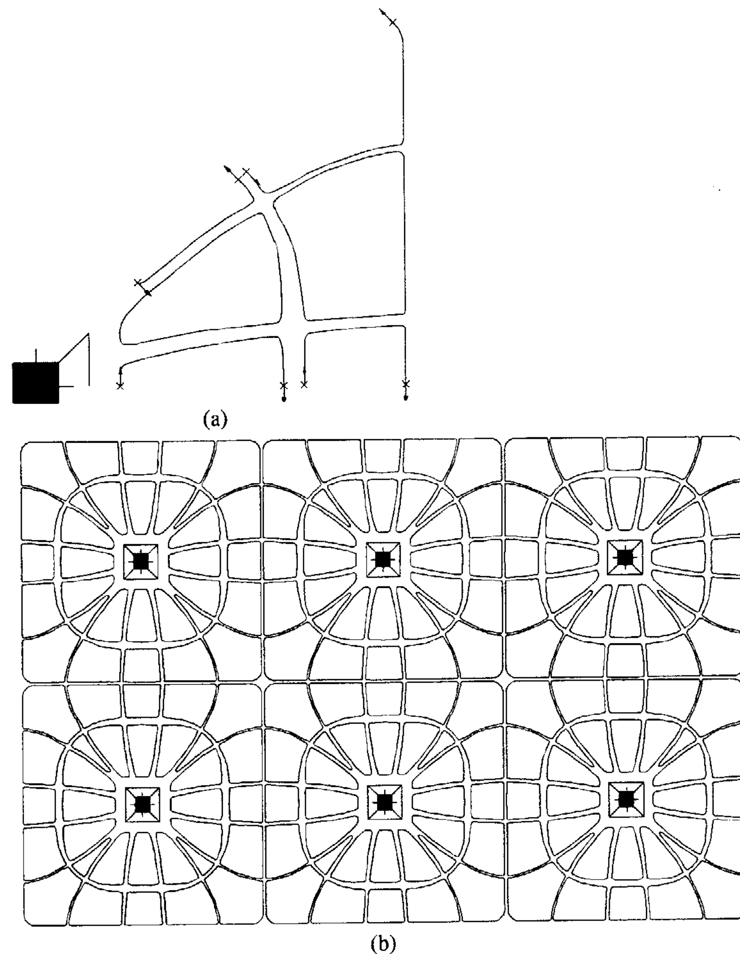


Figure 7.4 Concrete rib structure on the ceiling of a warehouse in Bologna, Italy, designed by engineer Pier Luigi Nervi: (a) the use of tangent lines at the boundary points of an octant drawn around a column and (b) the rib structure around six of the columns.

points (× marks) of the top polylines in figure 7.4a are set at $-45°$ and $135°$, respectively. Similarly, the tangents at the first and last vertex points on the bottom polylines are set at $\pm 90°$ to match slopes when the first quadrant is put into a polar array.

Another drawing with the AutoCAD circular-arc fit is shown in figure 7.5. The cane weave on the chair seat was drawn with the hatch pattern developed in section 3.4.

The Circular-Arc Curve-Fit Algorithm

To draw a pair of arcs between each vertex pair, you must insert a new vertex point between each pair of data points. The RP (Renner-Pochop) method specifies the positions of the inserted vertices. This method also prescribes the curve tangents at each of the data points.

Consider n data points providing the coordinates (x_i, y_i) of n vertices $\mathbf{V}_i$, where the subscript i takes on values from 0 through $n-1$. A polyline through these points has $n-1$ segments, numbered 0 through $n-2$; see figure 7.6.

The first step in the curve-fitting routine is the calculation of vectors tangent to the curve at each of the vertices. Although in AutoCAD the user has the option of specifying tangent directions, most often only the vertex points are given, with the algorithm determining the tangents. The RP method prescribes the tangent vector $\mathbf{t}_i$ at vertex i as a weighted average of the unit vectors $\mathbf{e}_{i-1}$ and $\mathbf{e}_i$ on either side of the vertex. Weighting factors w_0 and w_1 are functions of the segment lengths l_{i-1} and l_i, and a shape factor s, with the value $s = -1$ being used by the AutoCAD Fit algorithm. Additional weighting factors A and B for interior vertices depend on the unit vectors $\mathbf{e}_{i+1}$ and $\mathbf{e}_{i-2}$, so the tangents at these vertices depend on the four nearest polyline segments.

$$\mathbf{t}_i = Aw_0\mathbf{e}_{i-1} + Bw_1\mathbf{e}_i \tag{7.2}$$

where

$$w_0 = [(1 + s)l_{i-1} + (1 - s)l_i], \qquad w_1 = [(1 + s)l_i + (1 - s)l_{i-1}] \tag{7.3}$$

$$\mathbf{e}_{i-1} = \frac{1}{l_{i-1}} [l_{x,i-1}\,\mathbf{i} + l_{y,i-1}\mathbf{j}], \qquad \mathbf{e}_i = \frac{1}{l_i} [l_{x,i}\,\mathbf{i} + l_{y,i}\mathbf{j}] \tag{7.4}$$

Equation 7.2 holds for vertices with indices from $i = 1$ through $n-2$. For the two vertices adjacent to end points, $i = 1$ and $i = n-2$, the amplitude terms A and B each have the value 1. For the interior vertices, $i = 2$ through $n-3$, the amplitudes are computed by the following expressions.

$$A = |\mathbf{e}_{i+1} - \mathbf{e}_i|, \qquad B = |\mathbf{e}_{i-1} - \mathbf{e}_{i-2}| \tag{7.5}$$

where the expressions within the absolute value symbols can be readily evaluated by formulas for the unit vectors of the form given in equations 7.4.

The tangent vector $\mathbf{t}_0$ at the first vertex is given by

$$\begin{aligned} \mathbf{t}_0 &= 2(\mathbf{t}_1 \cdot \mathbf{e}_0)\mathbf{e}_0 - \mathbf{t}_1 \\ &= [w_0 + 2w_1\cos(\sigma_1 - \sigma_0)]\mathbf{e}_0 - w_1\mathbf{e}_1 \end{aligned} \tag{7.6}$$

where $\sigma_1 - \sigma_0$ is the angle between the directions of the first two polyline segments, obtained when equation 7.2 with $i = 1$ is used to expand the scalar product $\mathbf{t}_1 \cdot \mathbf{e}_0$. The tangent vector

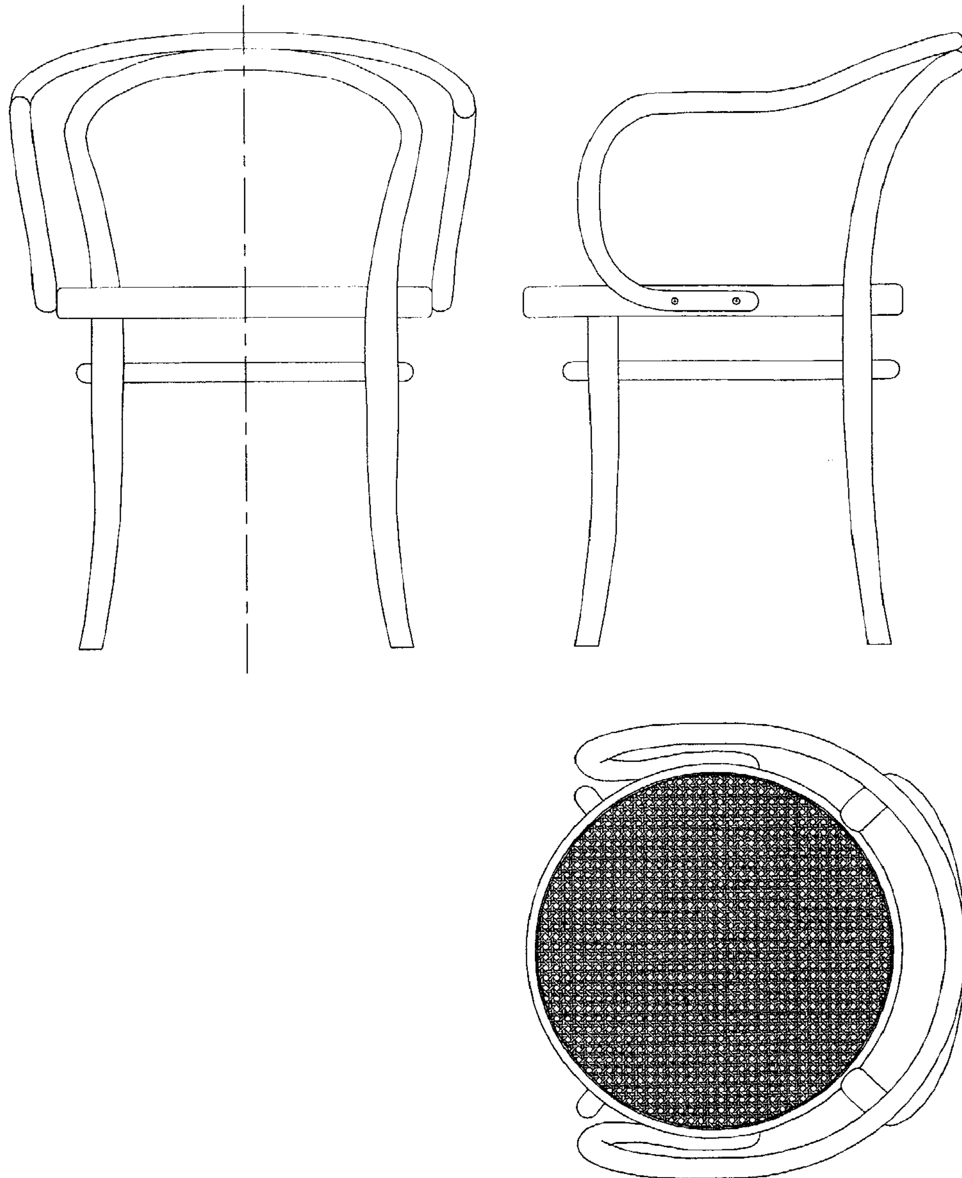


Figure 7.5 Bentwood armchair produced in 1870 by Gebrüder Thonet. Redrawn on AutoCAD from an illustration in *The Modern Chair: Classics in Production*, by Clement Meadmore, published by Van Nostrand Reinhold Co., 1975.

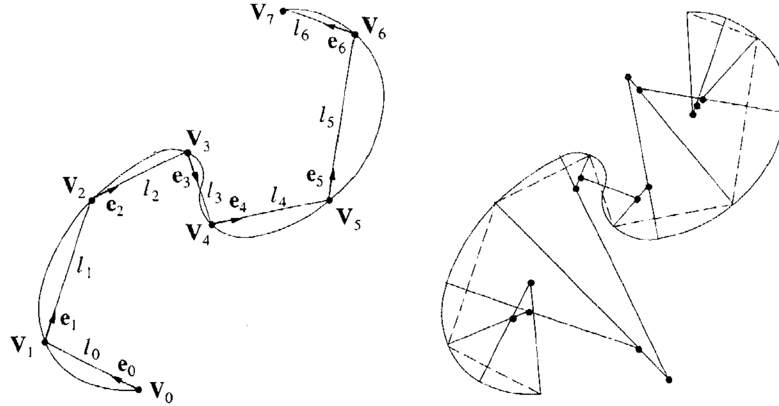


Figure 7.6 A polyline with eight vertex points $\mathbf{V}_0$ through $\mathbf{V}_7$. On the left, polyline segment lengths l_i and unit vectors $\mathbf{e}_i$ are shown, and on the right the arc centers and radii are drawn.

at vertex $i = n-1$ is calculated by a formula similar to that in equation 7.6, except with $\mathbf{e}_0$, $\mathbf{t}_0$, and $\mathbf{t}_1$ replaced by $\mathbf{e}_{n-2}$, $\mathbf{t}_{n-1}$, and $\mathbf{t}_{n-2}$.

The second step in the RP method is the calculation of the coordinates for the new vertices that are added between each pair of data points. Figure 7.7 shows three curve segments from figure 7.6, illustrating two different types of geometry. The first geometry, shown in 7.7a, arises when the tangent vectors $\mathbf{t}_i$ and $\mathbf{t}_{i+1}$ point to opposite sides of the segment (the figure shows a clockwise-turning polyline on the left and a counterclockwise polyline on the right). For this geometry the added point appears at the intersection of the lines that bisect the angles $2\alpha = \pm(\sigma_i - \phi_i)$ and $2\beta = \pm(\sigma_i - \phi_{i+1})$, which lie between the tangent vectors and the polyline segment (the + sign in these angle relations is for the clockwise case, and the − sign for the counterclockwise case).

As discussed in section 2.4, an intersection point may be readily calculated for two straight lines, which have equations of the form $y = mx + b$. The constants for the first line are determined by the fact that the line passes through the point $\mathbf{V}_i = (x_i, y_i)$ and has a slope m equal to $\tan(\sigma_i \pm \alpha) = \tan\{(\phi_i + \sigma_i)/2\}$, whereas the second line passes through $\mathbf{V}_{i+1} = (x_{i+1}, y_{i+1})$ and has a slope equal to $\tan(\phi_{i+1} \pm \beta) = \tan\{(\phi_{i+1} + \sigma_i)/2\}$. The intersection point for these two lines is obtained in the following form, after the tan() function is expressed as sin()/cos(), and multiplication by cos() is carried out to eliminate the divergences associated with the tan() functions when the argument is $\pm\pi/2$; that is, the following expressions hold for all orientations of the intersecting lines.

$$x_{int} = x_i + \frac{c_1[c_2(y_{i+1} - y_i) - s_2(x_{i+1} - x_i)]}{(s_1c_2 - s_2c_1)}$$
$$y_{int} = y_i + \frac{s_1[c_2(y_{i+1} - y_i) - s_2(x_{i+1} - x_i)]}{(s_1c_2 - s_2c_1)} \qquad (7.7)$$

where $c_1 = \cos\{(\phi_i + \sigma_i)/2\}$, $s_1 = \sin\{(\phi_i + \sigma_i)/2\}$, $c_2 = \cos\{(\phi_{i+1} + \sigma_i)/2\}$, and $s_2 = \sin\{(\phi_{i+1} + \sigma_i)/2\}$.

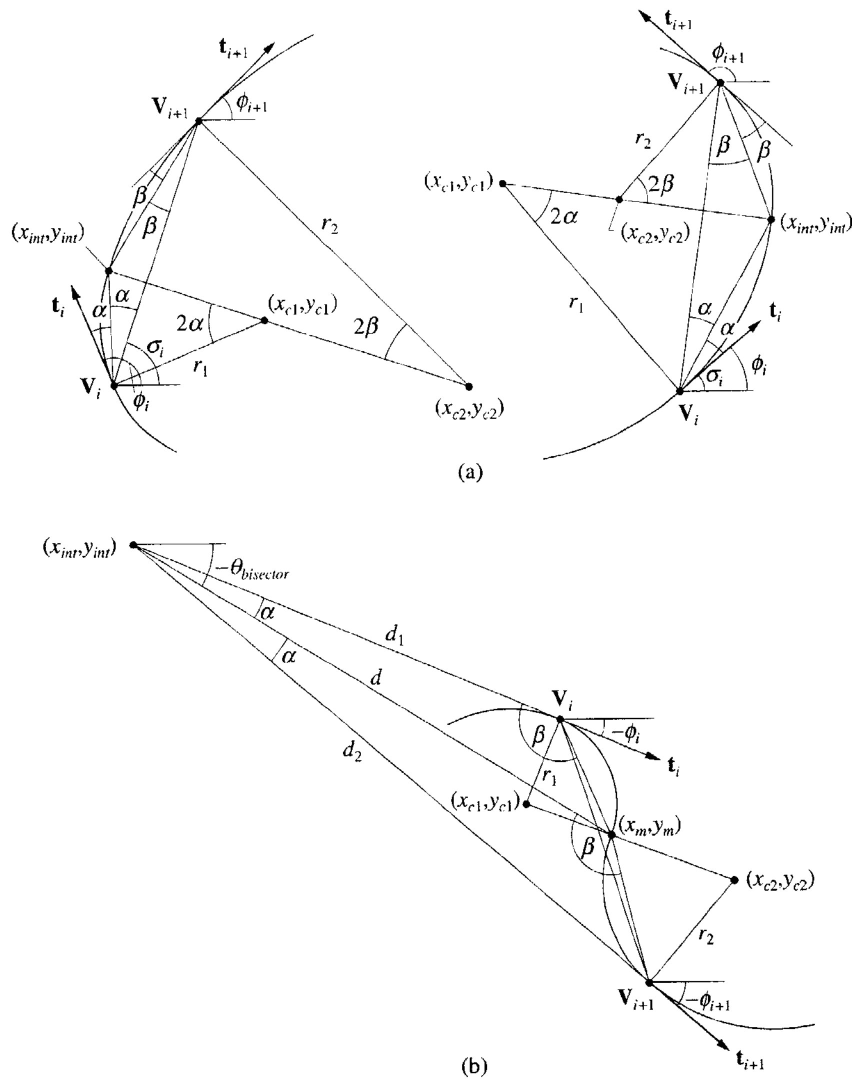


Figure 7.7 Geometry and nomenclature for calculating the coordinates of the added vertices and the arc centers. The case of tangent vectors pointing to opposite sides of the segment is shown in (a) with a clockwise-turning curve shown on the left, and a counterclockwise curve on the right. The diagram in (b) shows the case of the tangent vectors pointing to the same side of the segment, with the curve changing from clockwise to counterclockwise.

The arcs shown in figure 7.7a are subtended by angles 2α and 2β at the arc centers. Therefore, the right triangles in the figure provide the expressions

$$r_1 = \frac{d_1}{2|\sin \alpha|}, \qquad r_2 = \frac{d_2}{2|\sin \beta|} \tag{7.8}$$

where the variables d_1 and d_2 are the distances between the intersection point and the segment ends

$$d_1 = [(x_{int} - x_i)^2 + (y_{int} - y_i)^2]^{1/2}, \qquad d_2 = [(x_{int} - x_{i+1})^2 + (y_{int} - y_{i+1})^2]^{1/2} \tag{7.9}$$

Because the distances in equations 7.9 are positive, the computed radii are ensured positive values because absolute values of the sine functions appear in equations 7.8.

The vector of length r_1 between (x_i,y_i) and the arc center (x_{c1},y_{c1}) is perpendicular to the tangent vector $\mathbf{t}_i$, which lies in the direction ϕ_i to the horizontal. The vector of length r_2 between (x_{i+1},y_{i+1}) and the arc center (x_{c2},y_{c2}) is perpendicular to the tangent vector $\mathbf{t}_{i+1}$, which lies in the direction ϕ_{i+1}. Accordingly, the arc center coordinates may be calculated from the following relations.

$$\begin{aligned} x_{c1} &= x_i - ccw\, r_1 \sin\phi_i, & y_{c1} &= y_i + ccw\, r_1 \cos\phi_i \\ x_{c2} &= x_{i+1} - ccw\, r_2 \sin\phi_{i+1}, & y_{c2} &= y_{i+1} + ccw\, r_2 \cos\phi_{i+1} \end{aligned} \tag{7.10}$$

In equations 7.10, the multiplicative factor ccw takes on the value $+1$ for the counterclockwise case and the value -1 for the clockwise case.

The first arc is drawn from center (x_{c1},y_{c1}) with radius r_1, with the radius starting in a direction perpendicular to $\mathbf{t}_i$; that is, $\theta_{start} = \phi_i - ccw\, \pi/2$, where again the ccw factor takes into account the clockwise and counterclockwise orientations. The first arc ends at $\theta_{end} = \theta_{start} + 2\alpha$. The second arc starts at the angle on which the first angle ends, and is drawn to subtend an angle 2β at (x_{c2},y_{c2}).

The second case to be considered is the geometry shown in figure 7.7b, where the tangent vectors $\mathbf{t}_i$ and $\mathbf{t}_{i+1}$ point to the same side of the segment. The fitted curve along this segment will change orientation between clockwise and counterclockwise turning. The first step in calculating the position of the added vertex point is to determine the intersection point of lines extending along the tangent vectors at the segment ends. The coordinates (x_{int},y_{int}) of this point are computed from equations 7.7, but with the arguments of the cosine and sine terms being equal to the angles of the tangent vectors; that is, $c_1 = \cos\phi_i$; $s_1 = \sin\phi_i$, $c_2 = \cos\phi_{i+1}$, and $s_2 = \sin\phi_{i+1}$.

The added vertex point (x_m,y_m) is measured along $\theta_{bisector}$, which represents the bisector of the angle between the tangent vectors. Along this bisector there is only one appropriate distance d to the point (x_m,y_m) such that the arcs meet with the same slope, as shown in figure 7.7b. Reference to the geometry in this figure shows that the triangle formed by points (x_{int},y_{int}), (x_i,y_i), and (x_m,y_m) is similar to the triangle formed by (x_{int},y_{int}), (x_m,y_m), and (x_{i+1},y_{i+1}). Therefore the following equality exists between the ratio of triangle sides: $d/d_1 = d_2/d$. Thus, the distance d to the point (x_m,y_m) may be calculated from the equation

$$d = (d_1 d_2)^{1/2} \tag{7.11}$$

Although for some orientations the bisector angle, $\theta_{bisector}$, equals the average of the tangent angles, $(\phi_i+\phi_{i+1})/2$; for other orientations ϕ_i and/or ϕ_{i+1} must be increased by π. Instead of dealing separately with different orientations, one can alternatively calculate the bisector angle from the positions of the arc centers relative to the intersection point. The following result is obtained for the added vertex point coordinates.

$$x_m = x_{int} + d\cos\theta_{bisector}, \quad y_m = y_{int} + d\sin\theta_{bisector} \tag{7.12}$$

where

$$\theta_{bisector} = \frac{1}{2}\left[\tan^{-1}\left(\frac{y_i - y_{int}}{x_i - x_{int}}\right) + \tan^{-1}\left(\frac{y_{i+1} - y_{int}}{x_{i+1} - x_{int}}\right)\right] \tag{7.13}$$

The coordinates (x_{c1},y_{c1}) of the first arc center may be calculated from the top pair of equations 7.10. Because the turning orientation changes for the second arc (and the parameter *ccw* is determined by the first arc), the coordinates (x_{c2},y_{c2}) are computed from the bottom pair of equations in 7.10, but with the $-$ sign changed to $+$ in the x_{c2} equation and $+$ changed to $-$ in the y_{c2} equation.

The two arcs may draw with angles starting at the data points and ending at the inserted vertex at (x_m,y_m).

$$\text{First arc:} \quad \theta_{start} = \tan^{-1}\left(\frac{y_i - y_{c1}}{x_i - x_{c1}}\right), \quad \theta_{end} = \tan^{-1}\left(\frac{y_m - y_{c1}}{x_m - x_{c1}}\right)$$

$$\text{Second arc:} \quad \theta_{start} = \tan^{-1}\left(\frac{y_{i+1} - y_{c2}}{x_{i+1} - x_{c2}}\right), \quad \theta_{end} = \tan^{-1}\left(\frac{y_m - y_{c2}}{x_m - x_{c2}}\right) \tag{7.14}$$

The above equations, along with the function arc() from section 3.1, may be used to write a program subroutine arcFit() that fits two circular arcs between each pair of n data points. Because a number of different cases must be considered, the program subroutine arcFit() is rather long, so it is given in appendix B as listing 7.1.

7.3 PARAMETRIC REPRESENTATION OF CURVES

All of the curves described in the rest of this chapter have the same type of parametric representation. Consider the parameter $\bar{t}$, which starts with the value 0 at the vertex $\mathbf{V}_0$, increases to $\bar{t} = 1$ at vertex $\mathbf{V}_1$, then to $\bar{t} = 2$ at vertex $\mathbf{V}_2$, $\bar{t} = i$ at vertex $\mathbf{V}_i$, etc. With the variable $\bar{t}$ as the parameter, a point in two or three dimensions is presented as the row vector (matrix) $\mathbf{Q}_i(\bar{t})$. (Alternatively, column vectors may be used.)

$$\begin{aligned} \mathbf{Q}_i(\bar{t}) &= [x(\bar{t}) \quad y(\bar{t})] \quad \text{for} \quad \text{2-D} \\ \mathbf{Q}_i(\bar{t}) &= [x(\bar{t}) \quad y(\bar{t}) \quad z(\bar{t})] \quad \text{for} \quad \text{3-D} \end{aligned} \tag{7.15}$$

Parametric representation in terms of the variable $\bar{t}$ provides a convenient way to describe the curves, even when they have infinite slopes or turn back on themselves. Note

that a nonparametric representation such as $y = y(x)$ requires special treatment when the slope becomes infinite and when the curve doubles back to give multiple values of y for a given x-value.

The curve data points are called vertex points and are designated by the vector $\mathbf{V}_i$.

$$\begin{aligned} \mathbf{V}_i &= [x_i \quad y_i] \text{ for } \text{2-D} \\ \mathbf{V}_i &= [x_i \quad y_i \quad z_i] \text{ for } \text{3-D} \end{aligned} \tag{7.16}$$

Two-dimensional curves are illustrated in figure 7.8.

It is convenient to transform to a new independent parameter t, which varies between 0 and 1 as the curve is traversed between each set of vertices $\mathbf{V}_i$ and $\mathbf{V}_{i+1}$. An appropriate transformation is $t = \bar{t} - i$, where i is the number of the vertex at the start of the curve segment. This is a *uniform* parametric representation, because t varies over the same range between each pair of vertices. Uniform parametric representations are considered in sections 7.4–7.6, and nonuniform represention s in section 7.7.

Because vector equations are used, there is no difference in the formulation of the curve algorithms in two and three dimensions (this is an important benefit of the parametric representation). Although the following examples will be for two-dimensional cases, results for curves in three-dimensional space can be readily carried out by computing $z(t)$ in the same manner that the $x(t)$ and $y(t)$ coordinates are calculated.

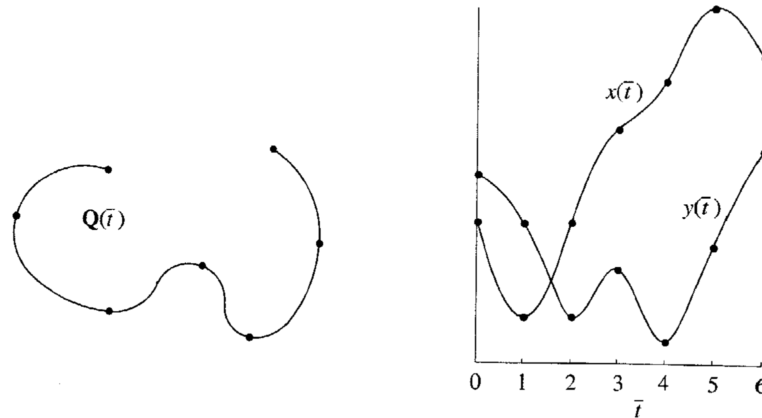


Figure 7.8 Parametric representation of a two-dimensional curve.

PARABOLIC BLEND

The curves developed in this section are produced by blending two parabolas, as illustrated in figure 7.9 (see Rogers and Adams [1990] for a more detailed derivation, and also for a generalized parabolic blend). The blended curve is shown as a dashed line between vertex points $\mathbf{V}_i$ and $\mathbf{V}_{i+1}$. The four vertices, $\mathbf{V}_{i-1}$ through $\mathbf{V}_{i+2}$, affect the shape of this dashed line by means of the formula

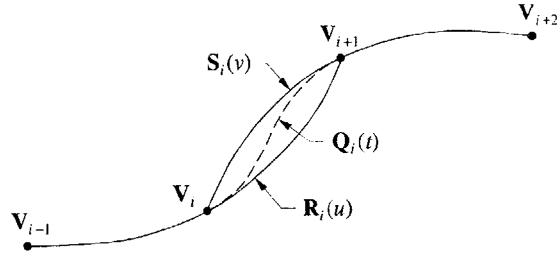


Figure 7.9 Configuration for parabolic blending.

$$Q_i(t) = V_{i-1}F_1(t) + V_iF_2(t) + V_{i+1}F_3(t) + V_{i+2}F_4(t)$$
$$= \sum_{k=-1}^{2} V_{i+k}F_{k+2}(t) \tag{7.17}$$

in which the variable t increases from 0 to 1 along the curve segment extending from the point V_i to the point V_{i+1}. That is, for this interpolation scheme, the overall curve comprises segments between the data points, which serve as joints or *knots* for the curve segments. In each curve segment the parameter t varies between 0 and 1.

In order to derive expressions for the functions F_1, F_2, F_3, and F_4 in equation 7.17, the individual parabolas shown in figure 7.9 are given the following parametric representations.

$$R_i(u) = [u^2 \quad u \quad 1][A]$$
$$S_i(v) = [v^2 \quad v \quad 1][B] \tag{7.18}$$

where $[u^2 \quad u \quad 1]$ represents a row vector (matrix) with three elements.

The first parabola is assumed to vary uniformly with the parameter u so that the points V_{i-1}, V_i, and V_{i+1} correspond to u values of 0, 1/2, and 1, respectively.

$$R_i(0) = V_{i-1} = [\,0 \quad 0 \quad 1][A]$$
$$R_i(1/2) = V_i = [1/4 \quad 1/2 \quad 1][A] \tag{7.19}$$
$$R_i(1) = V_{i+1} = [\,1 \quad 1 \quad 1][A]$$

The above equations can be written as a single matrix equation.

$$\begin{bmatrix} V_{i-1} \\ V_i \\ V_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1/4 & 1/2 & 1 \\ 1 & 1 & 1 \end{bmatrix} [A] \tag{7.20}$$

Inverting the first matrix on the right of equation 7.20 produces the following expression for [A]:

$$[A] = \begin{bmatrix} 2 & -4 & -2 \\ -3 & 4 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_{i-1} \\ V_i \\ V_{i+1} \end{bmatrix} \tag{7.21}$$

The same procedure is followed for $S_i(v)$, which crosses the points V_i, V_{i+1}, and V_{i+2} at the values $v = 0$, 1/2, and 1. The following value is obtained for the matrix [B] of equation 7.18.

$$[B] = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{V}_i \\ \mathbf{V}_{i+1} \\ \mathbf{V}_{i+2} \end{bmatrix} \tag{7.22}$$

The blended curve $\mathbf{Q}_i(t)$ is obtained from $\mathbf{R}_i(u)$ and $\mathbf{S}_i(v)$ by linear interpolation.

$$\mathbf{Q}_i(t) = (1-t)\mathbf{R}_i(u) + t\mathbf{S}_i(v) \tag{7.23}$$

The blended curve vector $\mathbf{Q}_i(t)$ appropriately reduces to $\mathbf{R}_i(u)$ when $t = 0$ and to $\mathbf{S}_i(v)$ when $t = 1$. That is, the blended curve will reach the values and the slopes of the individual parabolas at each end of the curve segment.

Equations 7.18, 7.21, and 7.22 can be used to substitute for $\mathbf{R}_i(u)$ and $\mathbf{S}_i(v)$ in equation 7.23. The variables u and v are assumed to be linear functions of t. The linear expressions below result when it is required that at $\mathbf{V}_i$, $t = 0$, $u = 1/2$, and $v = 0$; and at $\mathbf{V}_{i+1}$, $t = 1$, $u = 1$, and $v = 1/2$. (These values are consistent with the analysis above and with figure 7.9.)

$$u = \frac{1}{2}(1 + t), \qquad v = \frac{1}{2}t \tag{7.24}$$

Substitution of the expressions for $\mathbf{R}_i(u)$ and $\mathbf{S}_i(v)$ into equation 7.23 yields the following result for $\mathbf{Q}_i(t)$.

$$\begin{aligned} \mathbf{Q}_i(t) = {} & \begin{bmatrix} -\frac{t^3}{2} + t^2 - \frac{t}{2} & t^3 - t^2 - t + 1 & -\frac{t^3}{2} + \frac{t}{2} \end{bmatrix} \begin{bmatrix} \mathbf{V}_{i-1} \\ \mathbf{V}_i \\ \mathbf{V}_{i+1} \end{bmatrix} \\ & + \begin{bmatrix} \frac{t^3}{2} - \frac{3}{2}t^2 + t & -t^3 + 2t^2 & \frac{t^3}{2} - \frac{t^2}{2} \end{bmatrix} \begin{bmatrix} \mathbf{V}_i \\ \mathbf{V}_{i+1} \\ \mathbf{V}_{i+2} \end{bmatrix} \end{aligned} \tag{7.25}$$

Multiplying the matrices and collecting terms in equation 7.25 yields a solution of the form of equation 7.17, with the following expressions for the F terms.

$$\begin{aligned} F_1(t) &= -\frac{1}{2}(1 - t)^2 t \\ F_2(t) &= \frac{1}{2}(3t^3 - 5t^2 + 2) \\ F_3(t) &= \frac{1}{2}(-3t^2 + 4t + 1)t \\ F_4(t) &= \frac{1}{2}(t - 1)t^2 \end{aligned} \tag{7.26}$$

The above F functions are used for all of the interior curve segments. For the end segments of an open curve, the curve may be continued along the individual parabolas that lead to the end points. For the first line segment, equations 7.18 and 7.21 are used for the

parabola $\mathbf{R}_i(u)$, with a new t defined by $t = 2u$, so that t varies from 0 to 1 and u varies from 0 to 1/2 along the first curve segment. The F_4 function is zero along this first segment, with the other F functions describing the parabola

$$F_1(t) = \frac{1}{2}(1 - t)(2 - t)$$
$$F_2(t) = (2 - t)t \qquad (7.27)$$
$$F_3(t) = \frac{1}{2}(t - 1)t$$

In a similar manner, equations 7.18 and 7.22 are used for calculations along the parabola representing the last segment of the curve. Along this last segment, t is defined by $t = (2v - 1)$, so that t varies from 0 to 1 while v varies from 1/2 to 1. Along this last segment, F_1 is zero, and the remaining functions are

$$F_2(t) = \frac{1}{2}(t - 1)t$$
$$F_3(t) = 1 - t^2 \qquad (7.28)$$
$$F_4(t) = \frac{1}{2}(t + 1)t$$

The C-program subroutine listed below calculates an open-ended parabolic blend curve fixed by n points having coordinate values passed to the subroutine by the arrays x[] and y[]. The first *for* loop draws the curve for the first segment, the next double *for* loop draws the interior segments, and the last loop draws the last segment. In the *for* loops over the integer j, the parameter t is incremented by the operation $t += dt$ which equals the operation $t = t + dt$. The increment dt is determined by the parameter *nsegs*, which is the number of segments that make up the interval from $t = 0$ to $t = 1$.

```
void parBlend(int n,double x[],double y[])
{
    int     i,j,nsegs,xp,yp;
    double  t,dt,f1,f2,f3,f4;

    nsegs = 20; dt = 1./(double) nsegs;
    MoveTo((int) x[0],(int) y[0]);

    for (j=1,t=dt;j<=nsegs;j++,t+=dt)
    {
        f1 = .5*(1.-t)*(2.-t);
        f2 = (2.-t)*t;
        f3 = .5*(t-1.)*t;
        xp = (int) (f1*x[0]+f2*x[1]+f3*x[2]);
        yp = (int) (f1*y[0]+f2*y[1]+f3*y[2]);
```

```
        LineTo(xp,yp);
    }
    for (i=1;i<n-2;i++)
    {
        for (j=1,t=dt;j<=nsegs;j++,t+=dt)
        {
            f1 = -.5*(1.-t)*(1.-t)*t;
            f2 = .5*(3.*t*t*t-5.*t*t+2.);
            f3 = .5*(-3.*t*t+4.*t+1)*t;
            f4 = .5*(t-1.)*t*t;
            xp = (int) (f1*x[i-1]+f2*x[i]+f3*x[i+1]+f4*x[i+2]);
            yp = (int) (f1*y[i-1]+f2*y[i]+f3*y[i+1]+f4*y[i+2]);
            LineTo(xp,yp);
        }
    }
    for (j=1,t=dt;j<=nsegs;j++,t+=dt)
    {
        f2 = .5*(t-1.)*t;
        f3 = 1.-t*t;
        f4 = .5*(t+1.)*t;
        xp = (int) (f2*x[n-3]+f3*x[n-2]+f4*x[n-1]);
        yp = (int) (f2*y[n-3]+f3*y[n-2]+f4*y[n-1]);
        LineTo(xp,yp);
    }
}
```

A *closed* parabolic blend curve may be constructed by adding three points with indices n, $n+1$, and $n+2$, having the same coordinates as points 0, 1, and 2, respectively. The program for the closed curve contains only the interior double *for* loop with the index i extending from 1 to n. The closed curve is drawn starting at the second point, as required by the blended-curve algorithm, and continues to point $n+1$, which has the same coordinates as the second point. The additional point $n+2$ is needed because, as shown by equation 7.17 and the program listing above, the algorithm for interior curve segments requires four points that define segments before and after the line segment being drawn.

7.5 NATURAL CUBIC SPLINE

Parametric representation of the natural cubic spline follows equations 7.15. Again t varies from 0 to 1 along each curve segment. For the natural cubic spline a cubic polynomial interpolates the curve between the data points, which serve as the segment joints or knots. The cubic polynomial provides a sufficient number of coefficients to match the segment values, slopes (first derivatives), and curvatures (second derivatives) at the knots.

As illustrated in figure 7.10, the ith segment extends between vertices (data points) $\mathbf{V}_i$ and $\mathbf{V}_{i+1}$. Along this segment the curve is given by the formula

$$Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3 \tag{7.29}$$

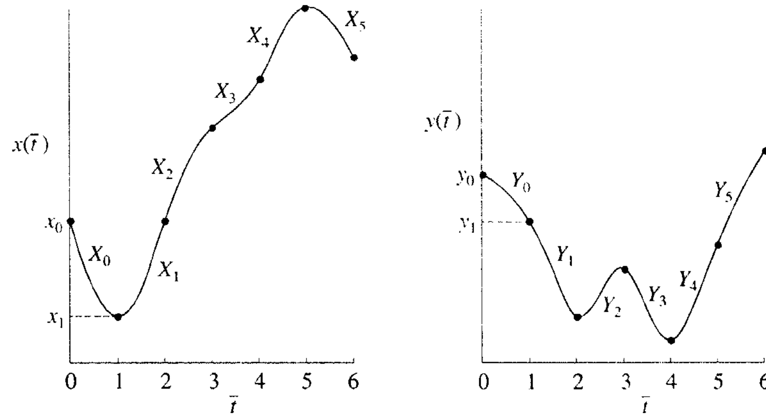


Figure 7.10 Notation for segments in a two-dimensional natural cubic spline curve.

where $Y_i(t)$ represents the $y(t)$ along the segment; a similar formula holds for $X_i(t)$ (and for $Z_i(t)$ in the 3-D case). Because t varies from 0 to 1 along each segment, $Y_i(t)$ reaches the ith vertex coordinate y_i when $t = 0$, and the vertex value y_{i+1} when $t = 1$. Similarly, $X_i(t) = x_i$ when $t = 0$ and $X_i(t) = x_{i+1}$ when $t = 1$; the coordinate values at the ends of the first segment are identified in figure 7.10.

The four unknown constants (a_i, b_i, c_i, and d_i) along each line segment will be determined by matching to the vertex values, requiring continuity of the first and second derivatives at the interior knots, and applying appropriate conditions at the curve ends. Along segment i, matching the vertex values y_i and y_{i+1} at the left and right ends of the segment gives the following equations.

$$Y_i(0) = y_i = a_i \tag{7.30}$$

$$Y_i(1) = y_{i+1} = a_i + b_i + c_i + d_i \tag{7.31}$$

The number of vertex points is n, with the segments numbered from 0 to $m = n - 1$. Matching vertex values provides 2 equations at each of the $m - 1$ interior knots plus 1 equation at each end of the total spline curve; therefore, matching values provides $2m$ equations involving the $4m$ unknown constants. Matching first and second derivatives at the $m - 1$ interior knots provides $2m - 2$ additional equations. The final two equations needed to solve for the $4m$ unknown constants are provided by requiring that the second derivatives vanish at the curve ends (this is the "natural" condition for a physical spline that is free at its ends).

Matching first derivatives at the interior knots provides $m - 1$ equations for the unknown constants. In solving for the unknown constants it proves useful to introduce new constants D_i representing the first derivatives at the vertex points. The first derivatives at the ends of segment i provide the equations

$$Y_i'(0) = D_i = b_i \tag{7.32}$$

$$Y_i'(1) = D_{i+1} = b_i + 2c_i + 3d_i \tag{7.33}$$

Equations 7.32 and 7.33 represent $2m$ equations; however, $m+1$ new constants D_i have been introduced, for a net gain equivalent to $m-1$ equations for the original constants.

Equation 7.30 solves for a_i, and equation 7.32 gives b_i in terms of D_i. Multiplying equation 7.31 by 3 and subtracting equation 7.33 leads to an expression for c_i, and 2 times equation 7.31 minus equation 7.33 provides an expression for d_i.

$$a_i = y_i \tag{7.34}$$

$$b_i = D_i \tag{7.35}$$

$$c_i = 3(y_{i+1} - y_i) - 2D_i - D_{i+1} \tag{7.36}$$

$$d_i = 2(y_i - y_{i+1}) + D_i + D_{i+1} \tag{7.37}$$

Matching the second derivatives at the $m-1$ interior knots, $Y_{i-1}''(1) = Y_i''(0)$, gives the following relation between the constants.

$$2c_{i-1} + 6d_{i-1} = 2c_i \tag{7.38}$$

After substitution from equations 7.36 and 7.37, the above equation reduces to the following expression for i values from 1 through $m-1$.

$$D_{i-1} + 4D_i + D_{i+1} = 3(y_{i+1} - y_{i-1}) \tag{7.39}$$

At the beginning of the curve the natural or free condition gives $Y_0''(0) = 2c_0 = 0$, which yields the following after equation 7.36 is used to substitute for c_0.

$$2D_0 + D_1 = 3(y_1 - y_0) \tag{7.40}$$

Similarly, the free boundary condition at the end of the curve gives $Y_{m-1}''(1) = 2c_{m-1} + 6d_{m-1} = 0$, which leads to the equation

$$D_{m-1} + 2D_m = 3(y_m - y_{m-1}) \tag{7.41}$$

The $m+1$ equations 7.39–7.41 for the $m+1$ first derivatives D_i may be put into matrix form (matrix operations are discussed in appendix C).

$$\begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & & \cdots & & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \cdot \\ D_{m-2} \\ D_{m-1} \\ D_m \end{bmatrix} = \begin{bmatrix} 3(y_1 - y_0) \\ 3(y_2 - y_0) \\ 3(y_3 - y_0) \\ \cdot \\ 3(y_{m-1} - y_{m-3}) \\ 3(y_m - y_{m-2}) \\ 3(y_m - y_{m-1}) \end{bmatrix} \tag{7.42}$$

The first matrix on the left is tridiagonal; that is, all elements are zero except for three elements in each row centered about the diagonal.

The above matrix equation is simplified by first multiplying the top row by the factor $\gamma_0 = 1/2$. Then the element to the left of the diagonal (the value 1) in each of the re-

maining rows is eliminated by subtracting the row directly above it multiplied by an appropriate factor γ_i. These operations produce the matrix equation

$$\begin{bmatrix} 1 & \gamma_0 & & & & & \\ & 1 & \gamma_1 & & & & \\ & & 1 & & & & \\ & & & \cdots & & & \\ & & & & 1 & \gamma_{m-2} & \\ & & & & & 1 & \gamma_{m-1} \\ & & & & & & 1 \end{bmatrix} \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \cdot \\ D_{m-2} \\ D_{m-1} \\ D_m \end{bmatrix} = \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \\ \cdot \\ \delta_{m-2} \\ \delta_{m-1} \\ \delta_m \end{bmatrix} \tag{7.43}$$

The parameters in equation 7.43 are given by the following relations.

$$\begin{aligned} \gamma_0 &= 1/2 \\ \gamma_i &= 1/(4 - \gamma_{i-1}) \quad \text{for } i = 1 \text{ to } m-1 \\ \gamma_m &= 1/(2 - \gamma_{m-1}) \end{aligned} \tag{7.44}$$

$$\begin{aligned} \delta_0 &= 3(y_1 - y_0)\gamma_0 \\ \delta_i &= [3(y_{i+1} - y_{i-1}) - \delta_{i-1}]\gamma_i \quad \text{for } i = 1 \text{ to } m-1 \\ \delta_m &= [3(y_m - y_{m-1}) - \delta_{m-1}]\gamma_m \end{aligned} \tag{7.45}$$

Equation 7.43 may be readily solved by starting with the last row and stepping backwards to the first row.

$$\begin{aligned} D_m &= \delta_m \\ D_i &= \delta_i - \gamma_i D_{i+1} \quad \text{for } i = m-1 \text{ to } 1 \end{aligned} \tag{7.46}$$

The above equations were used to write the subroutine natCubicSpline(), which is presented as listing 7.2 in appendix B. This subroutine plots a natural cubic spline interpolated between n data points having coordinates given by the arrays $x[]$ and $y[]$.

7.6 UNIFORM B-SPLINES

The simplest B-spline is the linear B-spline, which consists of straight lines drawn between the data points; that is, the linear B-spline is our standard polyline. The quadratic and cubic B-splines considered in this section are approximation curves rather than interpolation curves because these B-splines do not pass through the data points except at locations where there are multiple data points. The name B-spline arises from the basis (blending) functions used to describe the curves. At each point on a curve, the quadratic B-spline requires three basis functions, and the cubic B-spline requires four. These basis functions multiply the vertex point vectors in a manner similar to the $F_i(t)$ functions given in equation 7.17 for the parabolic blend.

Quadratic B-Spline

Figure 7.11a shows an open quadratic B-spline consisting of n vertex (control) points $\mathbf{V}_i$ and $n - 2$ curve segments $\mathbf{Q}_i(t)$. The tick marks delineate the interior knots joining curve segments, as well as the endpoints of the complete curve.

For the uniform B-splines considered in this section, the basis functions will have the same form along each curve segment. The open quadratic B-spline can be extended to the first and last control vertices by prescribing double points at these vertices. If these end-curve segments use the same basis functions as the interior segments, then the top end curves shown in figure 7.11b result. The bottom end curves result when the double end-points are treated as a nonuniform B-spline; see section 7.7. As will be shown later in this

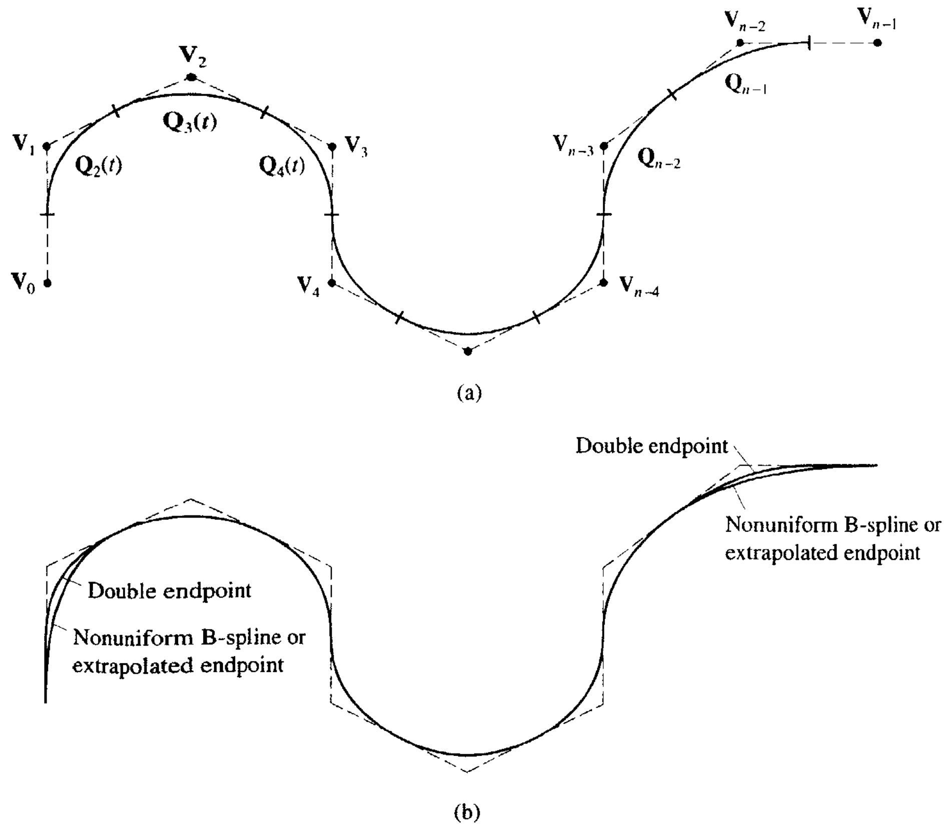


Figure 7.11 An open quadratic B-spline with the defining polyline (dashed): (a) curve for no special end conditions; (b) curve extended to the starting and ending control vertex points by means of two different procedures.

section, the bottom end curves can also be obtained by extrapolating the first and last control vertices to positions that relocate the curve ends to the original vertex positions.

The vector $\mathbf{Q}_i(t)$ for the coordinates along a quadratic B-spline may be expressed in terms of the three nearest vertex points $\mathbf{V}_{i-2}$, $\mathbf{V}_{i-1}$, and $\mathbf{V}_i$.

$$\begin{aligned}\mathbf{Q}_i(t) &= \mathbf{V}_{i-2}B_{i-2}(t) + \mathbf{V}_{i-1}B_{i-1}(t) + \mathbf{V}_iB_{i-0}(t) \\ &= \sum_{k=-2}^{0} \mathbf{V}_{i+k}B_{i+k}(t)\end{aligned} \tag{7.47}$$

The three basis functions contributing to the curve segment between knots i and $i+1$ are illustrated in figure 7.12a. The bell-shaped curves peak at the vertex points, and spread across exactly three segments. As in the previous curve formulations, the parameter t increases from 0 to 1 along the curve segment.

The basis functions are quadratic in the parameter t within each curve segment.

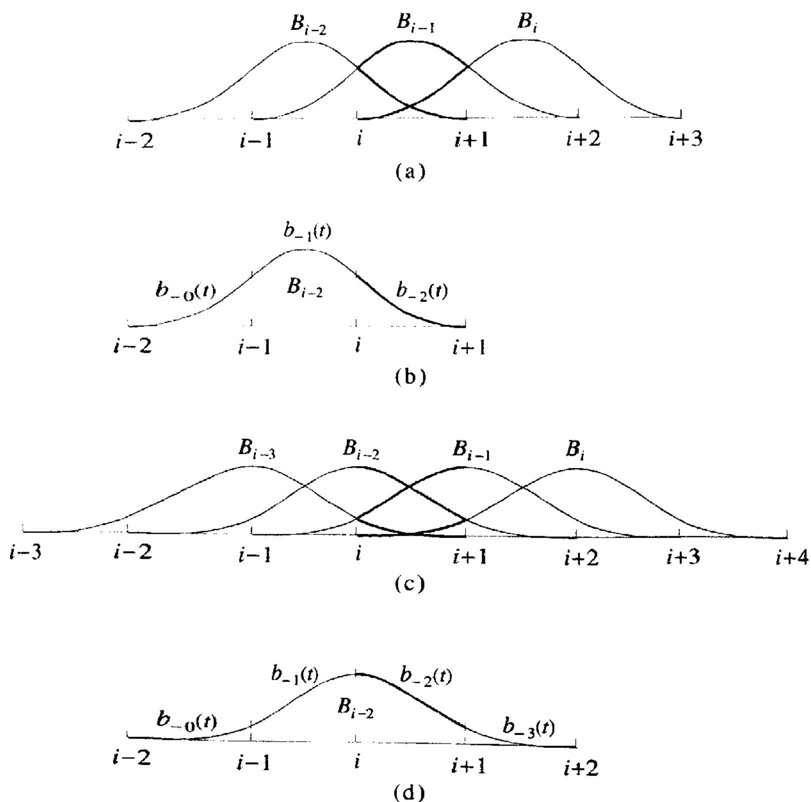


Figure 7.12 Basis functions for quadratic (a, b) and cubic (c, d) B-splines.

$$B_{i-2}(t) = a_j + b_j t + c_j t^2 \quad \text{for } i-2 \le j \le i \tag{7.48}$$

As illustrated in figures 7.12a and b, the basis function $B_{i-2}(t)$ extends across three segments, from knot $i-2$ to knot $i+1$, with t increasing from 0 to 1 along the curve from one knot to the next. Because the basis function coefficients are different in each curve segment, there are a total of 9 coefficients in equations 7.48. For the $B_{i-2}(t)$ curve shown in figure 7.12b, the quadratic polynomials 7.48 define a $b_i(t)$ function in each of the three segments.

$$\begin{aligned} b_{-0}(t) &= a_{i-2} + b_{i-2}t + c_{i-2}t^2 \\ b_{-1}(t) &= a_{i-1} + b_{i-1}t + c_{i-1}t^2 \\ b_{-2}(t) &= a_{i-0} + b_{i-0}t + c_{i-0}t^2 \end{aligned} \tag{7.49}$$

Because the shapes of the basis functions are the same, they may each be described as a composite of the three functions given in equations 7.49. Figure 7.12b shows that in the ith segment (between knots i and $i+1$), the basis function $B_{i-2}(t)$ contributes $b_{-2}(t)$, the function $B_{i-1}(t)$ contributes $b_{-1}(t)$, and $B_i(t)$ contributes $b_{-0}(t)$. Accordingly, equation 7.47 may be written as

$$\begin{aligned} \mathbf{Q}_i(t) &= \mathbf{V}_{i-2}b_{-2}(t) + \mathbf{V}_{i-1}b_{-1}(t) + \mathbf{V}_i b_{-0}(t) \\ &= \sum_{k=-2}^{0} \mathbf{V}_{i+k}b_k(t) \end{aligned} \tag{7.50}$$

The single basis function $B_{i-2}(t)$ will now be considered. Because the curve is a superposition of the basis functions having the same shape, continuity of $B_{i-2}(t)$ will ensure continuity of the curve. The continuity requirement that $B_{i-2}(t)$ and its first derivative be zero at its boundary knots $i-2$ and $i+1$ gives the relations

$$\begin{aligned} a_{i-2} &= 0 \\ a_i + b_i + c_i &= 0 \\ b_{i-2} &= 0 \\ b_i + 2c_i &= 0 \end{aligned} \tag{7.51}$$

The first and third equations of 7.51 correspond to knot $i-2$, where the parameter t has the value 0 because the knot is approached from the right. The second and fourth equations are for knot $i+1$, where t has the value 1.

At the interior knots, $i-1$ and i, the basis function and its first derivative must be continuous (the quadratic B-spline has continuity C^1). This continuity provides a total of four relations. Because the derivative of $B_{i-2}(t)$ is $b_j + 2c_j t$, the following equations result.

$$\begin{aligned} a_{i-2} + b_{i-2} + c_{i-2} &= a_{i-1} \\ a_{i-1} + b_{i-1} + c_{i-1} &= a_i \\ b_{i-2} + 2c_{i-2} &= b_{i-1} \\ b_{i-1} + 2c_{i-1} &= b_i \end{aligned} \tag{7.52}$$

The final equation needed to solve for the nine coefficients is provided by a normalization, which requires that the three basis functions contributing at any $t = 0$ sum to the value 1. Because each basis function has the same bell-shaped curve, normalization is provided by

$$b_{-0}(0) + b_{-1}(0) + b_{-2}(0) = 1$$

which gives (7.53)

$$a_{i-2} + a_{i-1} + a_i = 1$$

The nine equations 7.51, 7.52, and 7.53 are solved for the nine coefficients, which upon substitution into equations 7.49, yield

$$b_{-0}(t) = \frac{1}{2}t^2$$

$$b_{-1}(t) = \frac{1}{2}(1 + 2t - 2t^2) \qquad (7.54)$$

$$b_{-2}(t) = \frac{1}{2}(1 - t)^2$$

Equations 7.54 substituted into equation 7.50 represents a solution for the quadratic B-spline curve. Values for the nine coefficients may be obtained by comparing equations 7.49 with equations 7.54. These coefficients can readily be shown to satisfy equations 7.51, 7.52, and 7.53. Also, the resulting expressions 7.54 satisfy the following normalizing condition at all t values.

$$b_{-0}(t) + b_{-1}(t) + b_{-2}(t) = 1 \qquad (7.55)$$

Because three basis functions are needed to describe the curve, the curve starts with the second segment. For an *open* curve, as shown in figure 7.11a, the curve starts midway between the first two vertex points (since $b_{-0}(0) = 0$ and $b_{-1}(0) = b_{-2}(0) = 1/2$), and ends midway between the last two vertices. The open quadratic B-spline is drawn by subroutine quadBSpline() listed below.

In the subroutine quadBSpline(), the outer *for* loop over index i extends over the $nisegs = n - 2$ curve segments. In the inner *for* loop over the integer j, the parameter t is incremented by the operation $t+ = dt$, which equals the operation $t = t + dt$. The increment dt is set by the parameter *ntsegs,* which is the number of segments that make the interval from $t = 0$ to $t = 1$.

```
void quadBSpline(int n,double x[],double y[])
{
    int i,j,nisegs,ntsegs;
    double t,dt,b0,b1,b2,xp,yp;

    nisegs = n - 2;
    ntsegs = 20; dt = 1./(double) ntsegs;

    for (i=2; i<= nisegs+1; i++)
    {
```

```
            for (j=0,t=0.;j<=ntsegs;j++,t+=dt)
            {
                b0 = .5*t*t;
                b1 = .5*(1.+2.*t-2.*t*t);
                b2 = .5*(1.-t)*(1.-t);
                xp = b2*x[i-2]+b1*x[i-1]+b0*x[i];
                yp = b2*y[i-2]+b1*y[i-1]+b0*y[i];
                if (i==2 && j==0)
                    MoveTo((int) xp,(int) yp);
                else
                    LineTo((int) xp,(int) yp);
            }
        }
    }
```

The quadratic B-spline can be extended to the starting and ending control vertices by making these points double, that is, by placing one additional vertex at the location of the first vertex $\mathbf{V}_0$ and placing an additional vertex at the last location $\mathbf{V}_{n-1}$. Note that at $t = 0$ in the first segment, $b_{-0}(t) = 0$ and $b_{-2}(0)+b_{-1}(0) = 1$, therefore $\mathbf{Q}_i(t)$ in equation 7.50 will equal $\mathbf{V}_{i-1}$ if it is a double point (since $\mathbf{V}_{i-2}$ equals $\mathbf{V}_{i-1}$ at the double point). A similar analysis shows that the curve will extend to a double point at the end of the curve.

If the functions 7.54 are used for $b_{-0}(t)$, $b_{-1}(t)$, and $b_{-2}(t)$ in the first and last curve segments (with the first and last vertices being double points), then the curves in these segments differ somewhat from those drawn by AutoCAD. The AutoCAD curves correspond to treating the double points as a nonuniform B-spline, as discussed in section 7.7. The first and last curve segments can be drawn by using the appropriate nonuniform B-spline expressions for $b_{-0}(t)$, $b_{-1}(t)$, and $b_{-2}(t)$. But the same curve segments can be programmed more simply by an equivalent procedure, namely, by extrapolating the first and last vertices to new positions, without adding any new vertices.

The extrapolation procedure for the first vertex $\mathbf{V}_0$ consists of simply requiring that the new vertex position produce a curve that starts at the original vertex position. That is, using equation 7.50 at the beginning of the first curve segment ($i = 2$ and $t = 0$) produces the following condition:

$$x_0 = \frac{1}{2}x_0^{new} + \frac{1}{2}x_1, \qquad y_0 = \frac{1}{2}y_0^{new} + \frac{1}{2}y_1 \tag{7.56}$$

Equations 7.56 give the following expressions for the new position of the starting vertex.

$$x_0^{new} = 2x_0 - x_1, \qquad y_0^{new} = 2y_0 - y_1 \tag{7.57}$$

A similar extrapolation of the last vertex gives the expression

$$x_{n-1}^{new} = 2x_{n-1} - x_{n-2}, \qquad y_{n-1}^{new} = 2y_{n-1} - y_{n-2} \tag{7.58}$$

Equations 7.57–7.58 can be used to provide the following program lines at the beginning of the subroutine quadBSpline() to produce beginning and ending curve segments that match those drawn by AutoCAD.

```
x[0] = 2.*x[0]-x[1]; y[0] = 2.*y[0]-y[1];
x[n-1] = 2.*x[n-1]-x[n-2]; y[n-1] = 2.*y[n-1]-y[n-2];
```

A *closed* quadratic B-spline can be constructed from the subroutine quadBSpline() by adding two vertex points to form a periodic set of vertices. For example, vertices $\mathbf{V}_n = \mathbf{V}_0$ and $\mathbf{V}_{n+1} = \mathbf{V}_1$ can be added, with the number of curve segments increased to $nisegs = n$. Also, arrays x[] and y[] should be dimensioned to handle $n + 2$ values. The additional vertices are added by placing the following lines at the beginning of the subroutine quadBSpline().

```
x[n] = x[0]; y[n] = y[0];
x[n+1] = x[1]; y[n+1] = y[1];
```

B-spline curves can be drawn with AutoCAD by entering "S" for spline under the PEDIT command. The default spline is the cubic B-spline, but the quadratic spline can be drawn by changing the system variable SPLINETYPE from the value 6 (for the cubic B-spline) to the value 5. That is, enter SPLINETYPE, and then the number 5 (the command SETVAR may precede SPLINETYPE, but it is not necessary). Also, the following system variables can be changed: SPLFRAME can be changed from 0 to 1 to display the spline frame (polyline); SPLINESEGS may be increased from its default value of 8 to increase the number of segments used in drawing the spline curve.

Figure 7.13a shows some sample curves drawn with the quadratic B-spline option on AutoCAD. On the left of the figure, a vertical straight line appears because three sequential points were entered with the same x-values. In equation 7.50, the segment determined by these three points will have this same constant value of x because the $b_i(t)$ functions sum to the value 1. On the left of figure 7.13a it is also illustrated that when two points merge, the quadratic B-spline will pass through the double point, and the slope at the double point will be discontinuous.

The right side of figure 7.13a shows a closed quadratic B-spline drawn from four control points located at the corners of a square. When the upper-right-hand control point is changed to a double point, the curve passes through the double point and has a discontinuous slope at that point.

Cubic B-Spline

The cubic B-spline requires four basis functions at each point on the curve.

$$\begin{aligned} \mathbf{Q}_i(t) &= \mathbf{V}_{i-3}B_{i-3}(t) + \mathbf{V}_{i-2}B_{i-2}(t) + \mathbf{V}_{i-1}B_{i-1}(t) + \mathbf{V}_iB_{i-0}(t) \\ &= \sum_{k=-3}^{0} \mathbf{V}_{i+k}B_{i+k}(t) = \sum_{k=-3}^{0} \mathbf{V}_{i+k}b_k(t) \end{aligned} \tag{7.59}$$

The open cubic B-spline curve shown in figure 7.14a for n control points has $n - 3$ line segments $\mathbf{Q}_i(t)$ (one less segment than the quadratic B-spline shown for the same con-

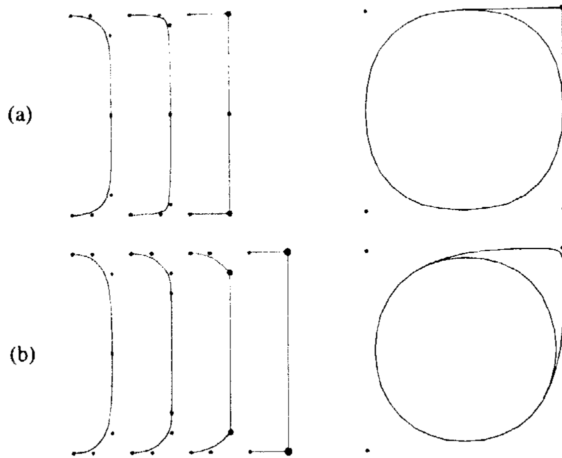


Figure 7–13 Sample B-spline curves drawn with AutoCAD: (a) the quadratic B-spline; (b) the cubic B-spline. The smallest dots represent single vertex points; the larger dots, double points; and the largest dots, triple points.

trol points in figure 7.11a). The index number i starts with $i = 3$ for the curve $\mathbf{Q}_i(t)$ in equation 7.59 because the first curve segment is controlled by the first four vertices: $\mathbf{V}_0$, $\mathbf{V}_1$, $\mathbf{V}_2$, and $\mathbf{V}_3$.

Figure 7.12c illustrates the four basis functions contributing to the curve segment between knots i and $i+1$. Each basis function extends across four curve segments, and is represented by a cubic polynomial in the parameter t. For example, the $B_{i-2}(t)$ function shown in figures 7.12c and d is given by the expression

$$B_{i-2}(t) = a_j + b_j t + c_j t^2 + d_j t^3 \quad \text{for } i - 2 \le j \le i + 1 \tag{7.60}$$

The four coefficients in each of the four segments spanned by a basis function represent a total of 16 constants to be determined. Because the equations are now cubic, continuity of second derivatives can be required. Accordingly, 16 equations for the coefficients are obtained by requiring the following: values, first derivatives, and second derivatives are zero at the boundary knots $i-2$ and $i+2$ (6 equations); values, first derivatives, and second derivatives are continuous at the three interior knots $i-1$, i, and $i+1$ (9 equations); and normalization is prescribed at $t = 0$ (1 equation).

The above procedure yields the following expressions.

$$\begin{aligned} b_{-0}(t) &= \frac{1}{6}t^3 \\ b_{-1}(t) &= \frac{1}{6}(1 + 3t + 3t^2 - 3t^3) \\ b_{-2}(t) &= \frac{1}{6}(4 - 6t^2 + 3t^3) \\ b_{-3}(t) &= \frac{1}{6}(1 - t)^3 \end{aligned} \tag{7.61}$$

Equations 7.61 satisfy the following normalization at each value of t.

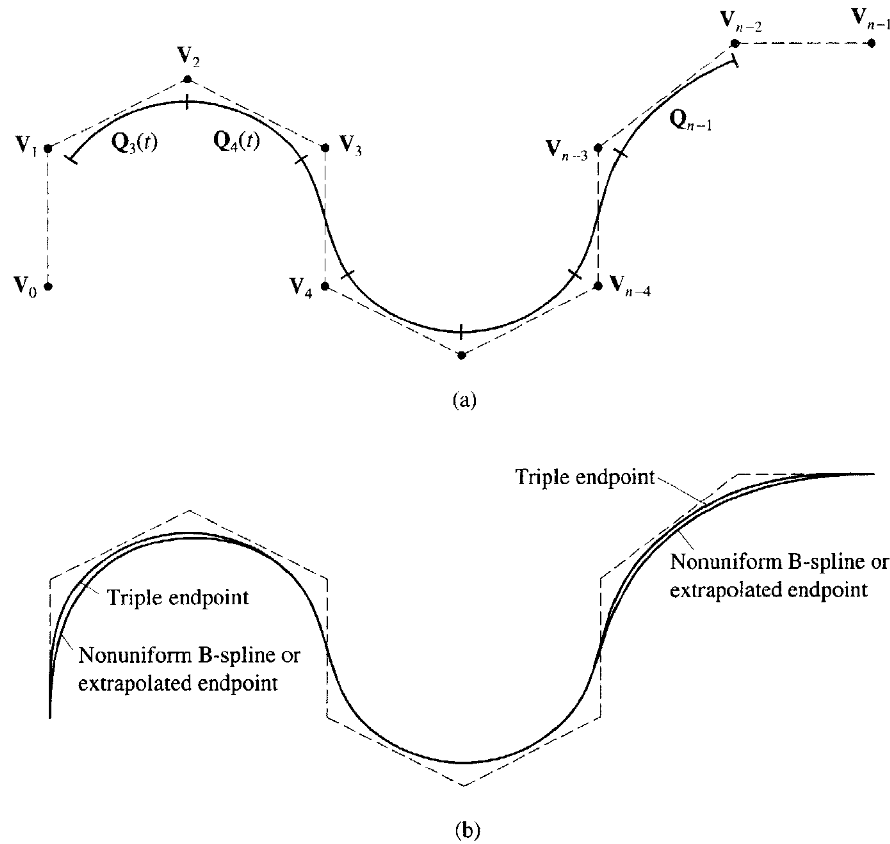


Figure 7.14 An open cubic B-spline with the defining polyline (dashed): (a) curve for no special end conditions; (b) curve extended to the starting and ending control vertex points by means of two different procedures.

$$b_{-0}(t) + b_{-1}(t) + b_{-2}(t) + b_{-3}(t) = 1 \tag{7.62}$$

In the open cubic B-spline subroutine given below, the outer *for* loop starts with index number $i = 3$ and extends to *nisegs*+2 so that a total number *nisegs* curve segments are drawn.

```
void cubicBSpline(int n,double x[],double y[])
{
    int i,j,nisegs,ntsegs;
    double t,dt,b0,b1,b2,b3,xp,yp;

    nisegs = n - 3;
    ntsegs = 20; dt = 1./(double) ntsegs;
```

```
        for (i=3; i<= nisegs+2; i++)
        {
            for (j=0,t=0.;j<=ntsegs;j++,t+=dt)
            {
                b0 = t*t*t/6.;
                b1 = (1.+3.*t+3.*t*t-3.*t*t*t)/6.;
                b2 = (4.-6.*t*t+3.*t*t*t)/6.;
                b3 = (1.-t)*(1.-t)*(1.-t)/6.;
                xp = b3*x[i-3]+b2*x[i-2]+b1*x[i-1]+b0*x[i];
                yp = b3*y[i-3]+b2*y[i-2]+b1*y[i-1]+b0*y[i];
                if (i==3 && j==0)
                    MoveTo((int) xp,(int) yp);
                else
                    LineTo((int) xp,(int) yp);
            }
        }
    }
```

The above subroutine will produce a cubic B-spline curve like the one shown in figure 7.14a. The curve can be extended to the starting and ending control vertices by making these points triple, that is, by placing two additional vertices at the location of the first vertex $\mathbf{V}_0$, and placing two additional vertices at the last location $\mathbf{V}_n$. When equations 7.61 are used for the $b_i(t)$ functions with each vertex in the triplets, the curves reach the starting and ending vertices as illustrated by the top curve in figure 7.14b. The bottom curve in 7.14b, which corresponds to the AutoCAD cubic spline, can be drawn by treating the triple points as a nonuniform B-spline (section 7.7), or by extrapolating vertices $\mathbf{V}_0$, $\mathbf{V}_1$, $\mathbf{V}_{n-1}$, and $\mathbf{V}_n$ in the following manner.

The following conditions are obtained from equation 7.59 by requiring that $\mathbf{V}_0$ be the starting point of the curve ($i = 3$ and $t = 0$ at the starting point).

$$x_0 = \frac{1}{6}x_0^{new} + \frac{2}{3}x_1^{new} + \frac{1}{6}x_2, \qquad y_0 = \frac{1}{6}y_0^{new} + \frac{2}{3}y_1^{new} + \frac{1}{6}y_2 \tag{7.63}$$

In equations 7.63 both the first and second vertices are moved to new positions. An additional condition to be imposed is that the slope of the curve at $\mathbf{V}_0$ be tangent to the line that connects the original vertices $\mathbf{V}_0$ and $\mathbf{V}_1$. This condition is imposed by the equations

$$\left(\frac{dx}{dt}\right)_{t=0} = -\frac{1}{2}x_0^{new} + \frac{1}{2}x_2 = \alpha(x_1 - x_0), \qquad \left(\frac{dy}{dt}\right)_{t=0} = -\frac{1}{2}y_0^{new} + \frac{1}{2}y_2 = \alpha(y_1 - y_0) \tag{7.64}$$

When the constant α in equation 7.64 is varied, different curves result, with $\alpha = 3$ giving curve segments that are identical to those drawn by AutoCAD and those given by the nonuniform B-spline. For $\alpha = 3$, equations 7.63 and 7.64 yield

$$x_0^{new} = x_2 - 6(x_1 - x_0), \qquad y_0^{new} = y_2 - 6(y_1 - y_0)$$

and

$$x_1^{new} = \frac{1}{2}(3x_1 - x_2), \qquad y_1^{new} = \frac{1}{2}(3y_1 - y_2) \tag{7.65}$$

Because the x_1^{new} and y_1^{new} values do not depend on x_0 and y_0, the x_0^{new} and y_0^{new} values can be calculated first, as shown in the following program lines. These lines may be inserted into the subroutine cubicBSpline() to duplicate the AutoCAD open cubic B-splines. Note that this extrapolation procedure requires at least five control points ($n \geq 5$).

```
x[0]   = x[2]+6.*(x[0]-x[1]); x[1] = 1.5*x[1]-.5*x[2];
y[0]   = y[2]+6.*(y[0]-y[1]); y[1] = 1.5*y[1]-.5*y[2];
x[n-1]  = x[n-3]+6.*(x[n-1]-x[n-2]);x[n-2] = 1.5*x[n-2]-.5*x[n-3];
y[n-1]  = y[n-3]+6.*(y[n-1]-y[n-2]);y[n-2] = 1.5*y[n-2]-.5*y[n-3];
```

A *closed* cubic B-spline can be constructed from the subroutine cubicBSpline() by adding three vertex points to form a periodic set of vertices. For example, vertices $\mathbf{V}_n = \mathbf{V}_0$, $\mathbf{V}_{n+1} = \mathbf{V}_1$, and $\mathbf{V}_{n+2} = \mathbf{V}_2$ can be added, with the number of curve segments increased to *nisegs* = n, so that the curve closes on itself. Also, arrays x[] and y[] should be dimensioned to handle $n + 3$ values.

The behavior of cubic B-spline curves is illustrated in figure 7.13b. The first curve on the left of figure 7.13b has three colinear vertex points, but no straight segment exists in the spline curve. Four colinear points in the next spline curve do produce a straight-line segment. This result may be deduced from equation 7.59 because the curve segment determined by these four colinear vertex points will have a constant x-value (the four $b_i(t)$ functions sum to the value 1). Figure 7.13b also illustrates the merging of vertices to form double points and triple points. Although the cubic B-spline approaches close to a double point, it does not pass through it, and the first derivative remains continuous (but the second derivative does not). The curve does pass through a triple point, at which it has a discontinuous first derivative.

Figure 7.15 shows that cubic B-splines produce curves of considerable smoothness and flexibility. These curves are noticeably smoother than curves drawn with the circular-arc (biarc) fit. The circular-arc fit tends to produce some small-scale extraneous wiggles as arcs are fit between data points. Note that the circular-arc curves in figure 7.5 do not appear to be as smooth as the cubic B-spline curves in figure 7.15.

Smooth curves can be drawn with considerable efficiency with the cubic B-spline. As an example, consider the character fonts displayed in figure 7.16. The top part of the figure shows the spline frame; that is, a polyline through the vertex points. Each character is constructed with two or three B-splines, plus some short straight lines that form the serifs at the ends of the splines. Vertical straight segments on the splines forming the letters *U* and *D* are constructed with four colinear vertices.

Fonts can also be constructed from quadratic curves. For example, the TrueType font format developed for the Macintosh is based on quadratic B-spline curves. More vertex points are required for quadratic curves, but a quadratic curve can be computed and printed faster. The PostScript printing language for laser printers uses cubic curves; namely, composite cubic Bézier curves. In section 7.8, Bézier curves of general order are discussed along with composite cubic Bézier curves.

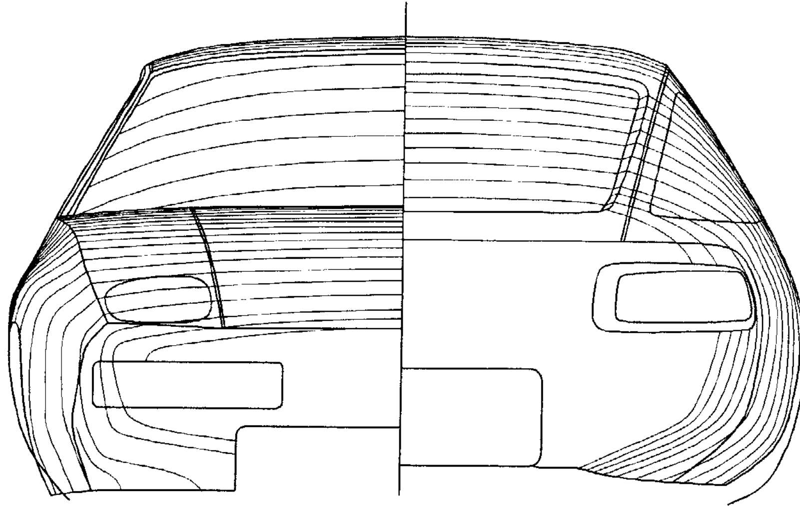


Figure 7.15 Vertical cross sections and some detail lines for a Porsche 928. These curves were drawn on AutoCAD with cubic B-splines.

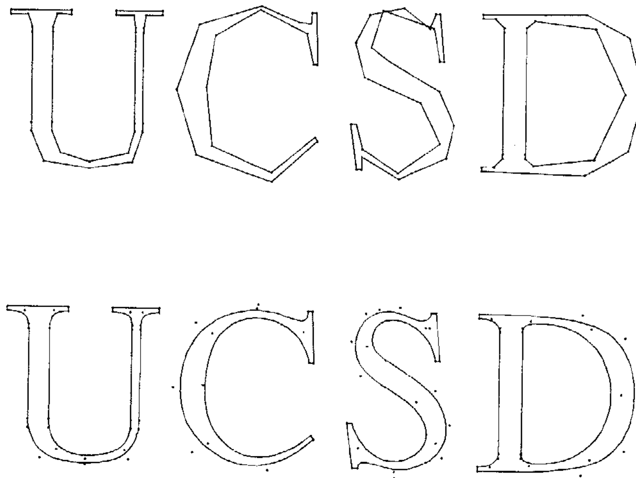


Figure 7.16 Characters formed by cubic B-splines, with the spline frame shown at the top. The dots identify the locations of the vertex points.

'.7 NONUNIFORM B-SPLINES

Nonuniform rational B-splines (NURBS) are generalizations of the uniform B-splines considered in the preceding section. NURBS curves and surfaces form the basis of a number of CAD systems because they can represent free-form shapes as well as common analytic shapes such as conic sections (circles, ellipses, parabolas, and hyperbolas) and primitive

surfaces like spheres, tori, cylinders, and cones. Although the basic AutoCAD release 12 software does not contain NURBS, AutoSurf™ release 2, which is an add-on package that works from within AutoCAD, is NURBS based.

The closed curves drawn in figure 7.17a show that the cubic uniform B-spline in a square polyline frame provides a rough approximation to a circle, while the quadratic uniform B-spline gives a poor approximation to a circle (the cubic B-spline is the inner solid curve, and the circles are shown with dashed lines). There is no way to accurately represent a circle with a uniform B-spline having a finite number of control vertices. On the other hand, NURBS curves have the flexibility to represent a continuous range of curves, as illustrated in figure 7.17b. For these NURBS curves, weights are assigned to each of the control vertices, which lie at the corners and side midpoints of the outer square. In figure 7.17b, the weights of the corner vertices are varied from 0 to ∞ to produce curves ranging from the inner rotated square to the outer square frame. When the weights of the corner vertices are $\sqrt{2}/2$ times the weights of the vertices on the square sides, the NURBS curve is a circle, as shown by the middle curve in figure 7.17b.

Before the theory of NURBS curves is developed in this section, nonuniform *nonrational* B-splines must be studied.

Nonuniform Nonrational B-Splines

The uniform B-splines described in the preceding section can be generalized by considering knots nonuniformly spaced along the parameter $\bar{t}$. Previously, the knots were considered to have equal spacing, with the $\bar{t}$ interval between knots being constant. The independent parameter t for the uniform B-splines described in section 7.6 varies between 0 and 1 and is related to the increasing parameter $\bar{t}$ by the relation $\bar{t} = i + t$, where i is the curve segment index number.

For convenience in the following discussion of nonuniform B-splines, the overbar is dropped from $\bar{t}$. The knots are placed along t as prescribed by the following knot vector $\mathbf{T}_{knot}$ having m elements t_0 through t_{m-1}.

$$\mathbf{T}_{knot} = [\underbrace{t_{min}, t_{min}, \ldots, t_{min}}_{k}, t_k, \ldots, t_{m-k-1}, \underbrace{t_{max}, t_{max}, \ldots, t_{max}}_{k}] \tag{7.66}$$

The knot vector $\mathbf{T}_{knot}$ in equation 7.66 is appropriate for an open nonuniform B-spline of order k and degree $k-1$. Thus, an open quadratic nonuniform B-spline has a knot vector with $k = 3$ identical elements at the start of the curve ($t_0 = t_1 = t_2 = t_{min}$, where t_{min} is of-

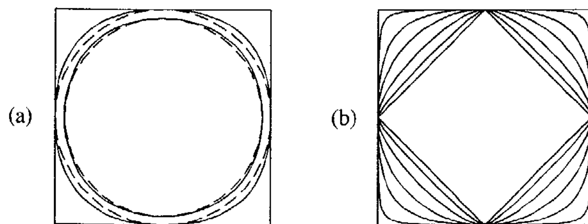


Figure 7.17 B-spline curves: (a) quadratic and cubic uniform B-splines, with control vertices at the corners of the square, and two circles (dashed lines); (b) quadratic nonuniform rational B-splines (NURBS), with the control vertices at the side midpoints having weights of 1.0, and the corner vertices having weights of 0.0, 0.1, 0.3, $\sqrt{2}/2$, 1.5, 5.0, and ∞.

ten set equal to 0) and $k = 3$ identical values at the curve end ($t_{m-3} = t_{m-2} = t_{m-1} = t_{max}$). The elements of the knot vector must be nondecreasing, $t_{min} \leq t_k$, $t_i \leq t_{i+1}$, and $t_{m-k-1} \leq t_{max}$.

For the open nonuniform B-spline, the following relation holds between the number of knots m, the number of control points n, and the spline order k.

$$m = n + k \tag{7.67}$$

Thus a quadratic spline ($k = 3$) will have three more knots than control points, and a cubic spline will have four more knots than control points.

AutoSurf can draw nonuniform B-splines with orders from 2 through 26. Because the order of the B-spline is limited by the number of control points, nine nonuniform B-splines (orders 2 through 10) can be drawn for the 10 control points shown in figure 7.18. AutoSurf draws free-form splines as nonrational B-splines but constructs primitive curves like the circle shown in figure 7.17 with NURBS.

Splines are drawn in AutoSurf with the SPLINE3D command and are edited with the EDITSP command. These commands offer some advanced features not present in the PLINE and PEDIT commands of AutoCAD. For example, with SPLINE3D, points on the curve may be entered, and AutoSurf will determine appropriate control point positions.

Closed splines are drawn in AutoSurf by selecting "closed" in the advanced options dialog box, and entering points on the curve. AutoSurf then computes appropriate control point positions, including starting and ending control points that lie midway between two other control points. Thus, AutoSurf uses an "open" spline to represent the closed curve, in the same manner that the circle is represented in the next subsection. The EDITSP command can be used to move the first or last control point to "open up" closed curves including circles (circles and ellipses are drawn with the ELLIPSE3D command).

The EDITSP command in AutoSurf moves and adds control points like the AutoCAD PEDIT command, but with EDITSP, points can be moved dynamically by using the mouse to move the cursor, and points on the curve (as well as control points) can be selected for displacement.

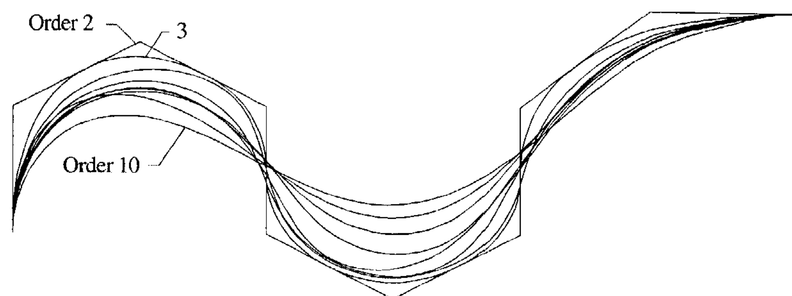


Figure 7.18 B-spline curves of orders 2 through 10 drawn with the AutoSurf SPLINE3D command. The control points are the same as those used in figures 7.11 and 7.14.

For the open nonuniform B-spline, the basis function of order k is represented with the notation $N_{i,k}(t)$, with the curve coordinates being represented by the vector $\mathbf{Q}(t)$.

$$\mathbf{Q}(t) = \sum_{i=0}^{n-1} \mathbf{V}_i N_{i,k}(t) \tag{7.68}$$

Although the form of equation 7.68 appears to be different from that of equations 7.47 and 7.59 for the uniform quadratic and cubic B-splines, the forms are comparable because for the quadratic case only three $N_{i,3}(t)$ functions will be nonzero along any curve segment, and only four $N_{i,4}(t)$ functions will be nonzero along a cubic B-spline curve segment. Thus the $N_{i,k}(t)$ functions for the nonuniform case correspond to the $B_i(t)$ functions of the uniform case.

The basis functions $N_{i,k}(t)$ of any order can be calculated from the Cox–deBoor recursion relation.

$$N_{i,k}(t) = \frac{(t - t_i)N_{i,k-1}(t)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - t)N_{i+1,k-1}(t)}{t_{i+k} - t_{i+1}} \tag{7.69}$$

To start the recursion relation 7.69, the following values are prescribed for the first-order basis functions.

$$N_{i,k}(t) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{7.70}$$

As illustrated in figure 7.19a, each first-order basis function spans the distance between adjacent knots.

Although equations 7.69–7.70 can be directly programmed to provide a spline of any order, it is instructive to develop solutions algebraically for orders 2 and 3. First, when using equation 7.69 to calculate $N_{i,2}(t)$, note that $N_{i,1}(t)$ is nonzero only in the region $t_i \le t < t_{i+1}$, and $N_{i+1,1}(t)$ is nonzero only in the region $t_{i+1} \le t < t_{i+2}$. Thus, equation 7.69 yields the following result.

$$N_{i,2}(t) = \frac{(t - t_i)}{t_{i+1} - t_i} \quad \text{for } t_i \le t < t_{i+1} \tag{7.71a}$$

$$N_{i,2}(t) = \frac{(t_{i+2} - t)}{t_{i+2} - t_{i+1}} \quad \text{for } t_{i+1} \le t < t_{i+2} \tag{7.71b}$$

The fact that the second-order basis function $N_{i,2}(t)$ spans across two curve segments (from knot t_i to t_{i+2}) is illustrated in figure 7.19a.

An expression for $N_{i+1,2}(t)$ may be obtained by transforming $i \to i + 1$ in equations 7.71. The basis function $N_{i,3}(t)$ is then calculated from equation 7.69 by requiring that only $N_{i,2}(t)$ is nonzero in the region $t_i \le t < t_{i+1}$, both $N_{i,2}(t)$ and $N_{i+1,2}(t)$ are nonzero in the region $t_{i+1} \le t < t_{i+2}$, and only $N_{i+1,2}(t)$ is nonzero in the region $t_{i+2} \le t < t_{i+3}$. The following result is obtained.

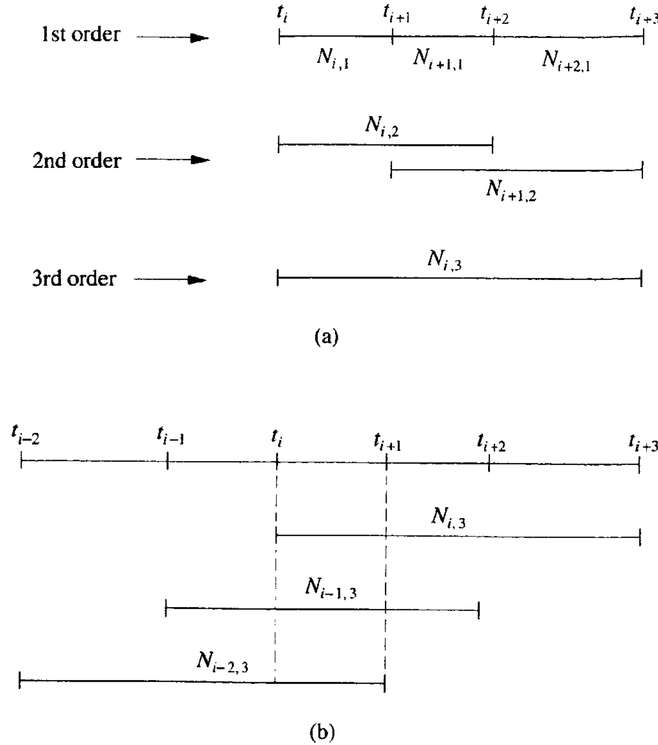


Figure 7.19 Basis function ranges for nonuniform B-splines: (a) the buildup of a third-order basis function from lower-order basis functions; (b) the three third-order basis functions that contribute to the B-spline in the region $t_i \le t \le t_{i+1}$.

$$N_{i,3}(t) = \frac{(t - t_i)^2}{(t_{i+2} - t_i)(t_{i+1} - t_i)} \quad \text{for } t_i \le t < t_{i+1} \tag{7.72a}$$

$$N_{i,3}(t) = \frac{(t - t_i)(t_{i+2} - t)}{(t_{i+2} - t_i)(t_{i+2} - t_{i+1})} + \frac{(t_{i+3} - t)(t - t_{i+1})}{(t_{i+3} - t_{i+1})(t_{i+2} - t_{i+1})} \quad \text{for } t_{i+1} \le t < t_{i+2} \tag{7.72b}$$

$$N_{i,3}(t) = \frac{(t_{i+3} - t)^2}{(t_{i+3} - t_{i+1})(t_{i+3} - t_{i+2})} \quad \text{for } t_{i+2} \le t < t_{i+3} \tag{7.72c}$$

The above procedure for calculating $N_{i,3}(t)$ is summarized in figure 7.19a. The first-order basis functions $N_{i,1}(t)$ and $N_{i+1,1}(t)$ are used to form $N_{i,2}(t)$ by equation 7.69; $N_{i+1,1}(t)$ and $N_{i+2,1}(t)$ are used to form $N_{i+1,2}(t)$; then $N_{i,2}(t)$ and $N_{i+1,2}(t)$ are used to form $N_{i,3}(t)$.

Because the second-order basis functions each span two curve segments, a general point t will be covered by two second-order basis functions; see figure 7.20a. As shown by figure 7.20a and equations 7.71, second-order basis functions are linear in t. This results in a second-order spline consisting of straight-line segments connecting the control points.

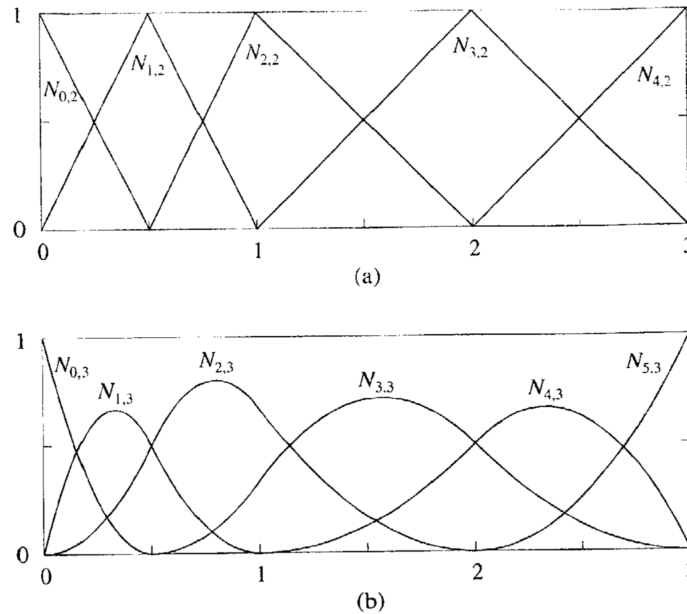


Figure 7.20 Basis functions for a nonuniform B-spline: (a) second-order basis functions for the knot vector $\mathbf{T}_{knot} = [\ 0, 0, 0.5, 1, 2, 3, 3\ ]$; (b) third-order basis functions for the same knot vector.

The third-order basis functions given by equations 7.72 cover three line segments, with the result that three third-order basis functions contribute at any t value. Third-order basis functions are shown in figure 7.20b for a particular nonuniform knot vector.

Figure 7.19b shows the three third-order basis functions that contribute for a value of t between knots t_i and t_{i+1}. The first is $N_{i,3}(t)$, which contributes a term (here called $b_{-0}(t)$ in analogy to the uniform B-spline case) determined by equation 7.72a; the second function is $N_{i-1,3}(t)$, which contributes a term $b_{-1}(t)$ calculated from 7.72b with i replaced by $i - 1$; and the third is $N_{i-2,3}(t)$, which contributes a term $b_{-2}(t)$ calculated from 7.72c with i replaced by $i - 2$. Thus, the curve coordinates along segment i may be calculated with a formula of the same form as for the uniform case in equation 7.50. Accordingly, the subroutine nonuQuadBSpln() given below is the same as the uniform quadratic B-spline subroutine quadBSpline(), except that a knot vector passed to the subroutine is used to calculate the $b_i(t)$ functions for the nonuniform B-spline.

```
void nonuQuadBSpln(int n,double x[],double y[],double tknot[])
{
    int i,j,nisegs,ntsegs;
    double t,dt,b0,b1,b2,xp,yp;
    double tn1,t0,t1,t2;

    nisegs = n - 2; ntsegs = 20;
```

```
    for (i=2; i<=nisegs+1; i++)
    {
        tn1 = tknot[i-1]; t0 = tknot[i];
        t1 = tknot[i+1]; t2 = tknot[i+2];
        if (t1 > t0)
        {
            dt = (t1-t0)/(double) ntsegs;
            for (j=0,t=t0;j<=ntsegs;j++,t+=dt)
            {
                b0 = (t-t0)*(t-t0)/((t2-t0)*(t1-t0));
                b1 = (t1-t)*(t-tn1)/((t1-t0)*(t1-tn1)) +
                        (t2-t)*(t-t0)/((t2-t0)*(t1-t0));
                b2 = (t1-t)*(t1-t)/((t1-tn1)*(t1-t0));
                xp = b2*x[i-2]+b1*x[i-1]+b0*x[i];
                yp = b2*y[i-2]+b1*y[i-1]+b0*y[i];
                if (j == 0)
                    MoveTo((int) xp, (int) yp);
                else
                    LineTo((int) xp, (int) yp);
            }
        }
    }
}
```

In the subroutine nonuQuadBSpln( ), knots are related to the index i by the following notation: $tn1$ stands for "i negative 1", or knot number $i-1$; $t0$ for knot $i+0$; $t1$ for $i+1$; and $t2$ for $i+2$. Note that the first knot, $tknot[0]$, does not appear in the subroutine. Although $tknot[0] = t_0$ appears in equations 7.72a and b that represent $N_{0,3}(t)$, only equation 7.72c is used to represent $N_{0,3}(t)$ in the first curve segment. In a similar manner, the last knot is not used directly in the curve calculation.

As illustrated in figure 7.20b, the identical values (0) for the second and third knots produce the following basis functions at $t = 0$: $N_{0,3}(0) = 1$, $N_{1,3}(0) = 0$, and $N_{2,3}(0) = 0$. These values produce a curve that starts at the first control point. Similarly, the multiple knot values at the end of the knot vector force the curve to extend to the last control point.

When the knots are uniformly placed, the third-order basis functions $N_{i,3}(t)$ over interior line segments reduce to the uniform B-spline basis functions $B_i(t)$. That is, the three functions in equations 7.72a–c reduce to the three quadratic polynomials $b_i(t)$ given in equation 7.54 when the parameter t is replaced by $i + t$, where the new t varies from 0 to 1.

When t is replaced by $i + t$, the first curve segment is covered by parts of three basis functions: $N_{0,3}(t) \equiv b_{-0}(t)$ calculated by equation 7.72a with $i = 2$, $N_{1,3}(t) \equiv b_{-1}(t)$ calculated by equation 7.72b with $i = 1$, and $N_{2,3}(t) \equiv b_{-2}(t)$ calculated by equation 7.72c with $i = 0$. (Or equivalently, the value $i = 2$ for the first curve segment can be substituted in the formulas in the nonuQuadBSpln() subroutine.) This procedure was used to derive the following relations.

$$b_{-0}(t) = \frac{1}{2}t^2$$
$$b_{-1}(t) = \frac{1}{2}t\,(4 - 3t) \qquad (7.73)$$
$$b_{-2}(t) = (1 - t)^2$$

In a similar manner, when t is replaced by $i + t$, the last curve segment yields the functions

$$b_{-0}(t) = t^2$$
$$b_{-1}(t) = \frac{1}{2}(1 + 2t - 3t^2) \qquad (7.74)$$
$$b_{-2}(t) = \frac{1}{2}(1 - t)^2$$

The AutoCAD quadratic B-spline curves are reproduced when equations 7.73 are used for the first curve segment, equations 7.54 for the interior segments, and equations 7.74 for the last segment. As discussed in section 7.6 on uniform B-splines, equations 7.54 can be used with an extrapolation procedure on the first and last control points to produce the same first and last curve segments as equations 7.73 and 7.74.

The cubic (fourth order) nonuniform B-spline can be calculated by the same procedures as described above for the quadratic case. For the cubic case, four basis functions determine a portion of the curve. These fourth-order basis functions can be calculated from the third-order functions 7.72 by using the Cox–deBoor formula 7.69. (The implementation of the fourth-order nonuniform B-spline is left for exercises 7.17–7.19). When the knots are uniformly spaced, the uniform cubic B-spline discussed in section 7.6 is recovered.

Nonuniform Rational B-Splines (NURBS)

The term *rational* in NURBS signifies that a ratio of basis functions is formed. Specifically, the ratio is formed by dividing by the sum of the basis functions. Also, a weight w_i is assigned to each vertex. Thus, equation 7.68 is replaced by the following.

$$\mathbf{Q}(t) = \frac{\sum_{i=0}^{n-1} \mathbf{V}_i w_i N_{i,k}(t)}{\sum_{i=0}^{n-1} w_i N_{i,k}(t)} = \sum_{i=0}^{n-1} \mathbf{V}_i R_{i,k}(t) \qquad (7.75)$$

In equation 7.75, the rational B-spline basis functions $R_{i,k}(t)$ are defined by

$$R_{i,k}(t) = \frac{w_i N_{i,k}(t)}{\sum_{i=0}^{n-1} w_i N_{i,k}(t)} \qquad (7.76)$$

The weights w_i are often represented by the variable h_i because they can be interpreted geometrically in terms of homogeneous coordinates; see, for example, Rogers and Adams (1990), or Piegl (1991).

A subroutine for nonrational B-splines can be developed by passing to the subroutine an array *w*[] for the vertex weights and by using these weights in the calculation of the curve coordinates *xp* and *yp*. In particular, the subroutine nonuQuadBSpln() above can be modified for rational B-splines by replacing the two lines for *xp* and *yp* by the following three program lines, where the variable *bsum* is assigned a double data type.

```
bsum = b2*w[i-2]+b1*w[i-1]+b0*w[i];
xp = (b2*w[i-2]*x[i-2]+b1*w[i-1]*x[i-1]+b0*w[i]*x[i])/bsum;
yp = (b2*w[i-2]*y[i-2]+b1*w[i-1]*y[i-1]+b0*w[i]*y[i])/bsum;
```

For the closed curves shown in figure 7.17b, there are nine control points starting and ending at the midpoint of one of the sides of the outer square. The array for the weights has unit values at the side midpoints; that is, $w[0] = w[2] = w[4] = w[6] = w[8] = 1.0$, and the corner vertices are assigned equal weights: $w[1] = w[3] = w[5] = w[7] = w_c$, where the seven closed curves from the inner rotated square to the outer square are formed with w_c values of 0.0, 0.1, 0.3, $\sqrt{2}/2$, 1.5, 5.0, and ∞. A circle is formed for the value $w_c = \sqrt{2}/2$.

The curves of figure 7.17b are separated and replotted in figure 7.21, along with plots of the basis functions. The curve is an open curve that starts at one of the side control points (whose locations are shown on the square at the lower right of figure 7.21). Because of the double knot values, the curve always passes through the control points located at the side midpoints. When the corner control vertices have zero weight ($w_c = 0$), the curve reduces to straight line segments between the side control points. At the other limit, when $w_c \to \infty$, the curve extends all the way to the polyline frame defined by the control vertices.

In terms of the basis functions shown on the left of figure 7.21, the double knot values force one of the basis functions to peak at the value 1 (and the others to drop to 0) at each knot location ($t = 0, 1, 2, 3,$ and 4). These peaks force the curve to start at one of the side control points, then pass through the other three side control points, and end on the first side control point. The curves shown in figure 7.21 start on the control point at the middle of the bottom side of the square because the control points were given by the following vectors: $x[] = [1, 2, 2, 2, 1, 0, 0, 0, 1]$ and $y[] = [0, 0, 1, 2, 2, 2, 1, 0, 0]$.

Circles drawn on AutoSurf with the ELLIPSE3D command are formed as open B-splines, like the curves described above. When the EDITSP command is used to display control points, the square frame is shown, and the selection of control points starts and ends with a point at the middle of one side of the frame, indicating an open curve. Furthermore, if the starting or ending control point is moved (with the Pull option), displacement of the first or last control point opens up the curve and spline frame.

Alternatively, a circle could be represented by four 90° circular arcs, each drawn with three control points. For example, the lower-right quarter circle can be drawn with the following knot vector, weight vector, and control point coordinate vectors: $\mathbf{T}_{knot} = [\,0, 0, 0, 1, 1, 1\,]$, $\mathbf{w} = [\,1, \sqrt{2}/2, 1]$, $x[] = [1, 2, 2]$, and $y[] = [0, 0, 1]$.

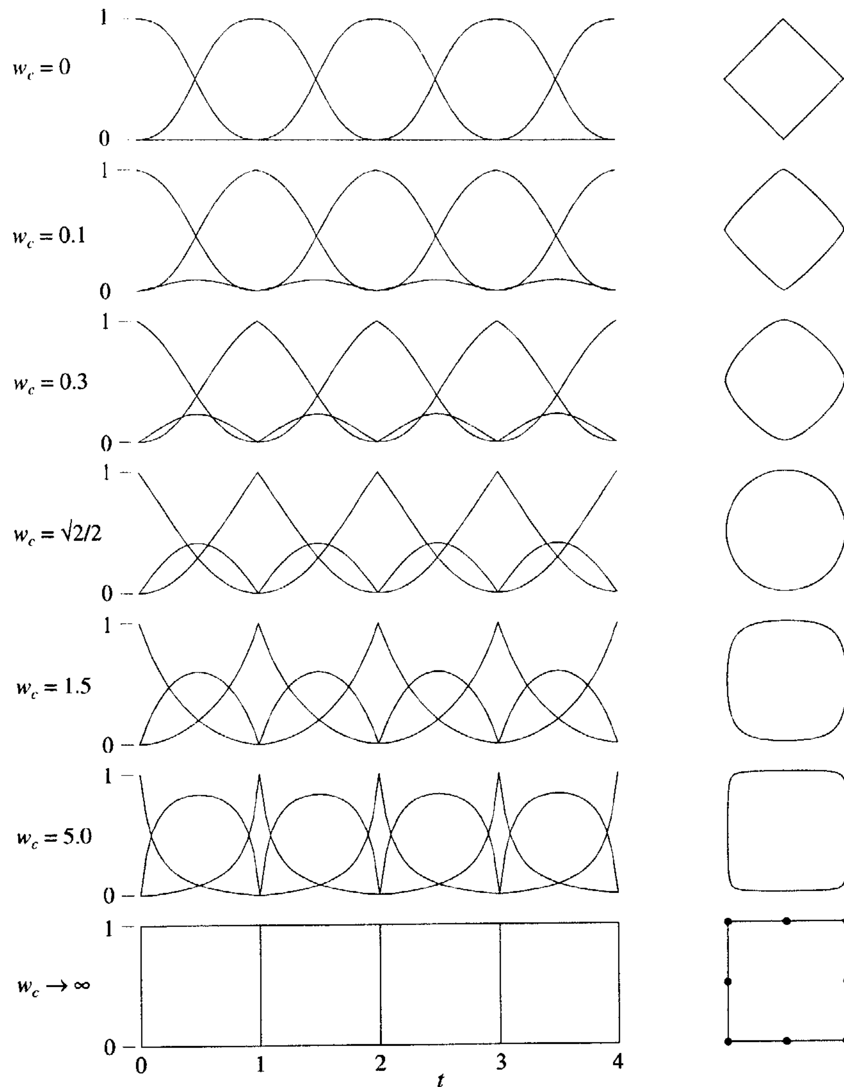


Figure 7.21 NURBS basis functions on the left, with the resulting closed curves on the right, calculated with the knot vector $\mathbf{T}_{knot}$ = [0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4]. The control vertices lie on the side midpoints and on the corners of the square shown on the lower right. The side vertices have a unit weight factor, and the corner vertices a weight of w_c, according to the weight vector $\mathbf{w}$ = [1, w_c, 1, w_c, 1, w_c, 1, w_c, 1].

A circular arc of any angle can be represented by a quadratic NURBS curve; see, for example, Rogers and Adams (1990). Three arcs of 120° each can form a circle, and combining these arcs into a single curve leads to the circle representation given in exercise 7.22.

.8 BÉZIER CURVES

Two types of Bézier curves are considered below: first, the general Bézier curve of order $m = n - 1$, where n is the number of data points. This curve will start at the first vertex (data) point and extend to the last point, with the intermediate vertex points acting as control points. Each point on the curve is controlled by all of the vertex points, with the result that vertex points have very little local control of the curve when the Bézier curve is of high order. Although AutoCAD does not incorporate single Bézier curves, it does use Bézier surfaces defined by a mesh of Bézier curves of order determined by the number of data points. A discussion of Bézier surfaces is included in chapter 12.

The second type to be considered is a composite cubic Bézier curve, which is made up of cubic Bézier curves connected at inserted points. Such composite cubic Bézier curves are used to designate font shapes in PostScript software, which is widely used to produce high-quality characters on laser printers and typesetters.

General-Order Bézier Curve

The nonuniform B-spline curve reduces to the special case of a Bézier curve when the order $k = n$, where n is the number of vertices in the control polyline. The total number of knots given by equation 7.67 is $m = 2k = 2n$. Using the values $t_{min} = 0$ and $t_{max} = 1$, the knot vector for a Bézier curve is given by the equation

$$\mathbf{T}_{knot} = [\underbrace{0, 0, ..., 0,}_{k} \underbrace{1, 1, ..., 1,}_{k}] \tag{7.77}$$

The vector $\mathbf{Q}(t)$ for points along the curve is thus given by

$$\mathbf{Q}(t) = \sum_{i=0}^{d} \mathbf{V}_i P_{i,d}(t) \tag{7.78}$$

where d is the degree of the curve, which is related to the curve order k by the relation $d = k - 1 = n - 1$. The entire curve is drawn as t varies from 0 to 1.

Equation 7.78 is the same as equation 7.68, except that $P_{i,d}(t)$ has been used to denote the Bézier basis functions, which equal the nonuniform B-spline basis functions for the case $k = n$. The Bézier basis functions are given by the following Bernstein polynomials of order d.

$$P_{i,d}(t) = \binom{d}{i} t^i (1 - t)^{d-i}, \qquad \binom{d}{i} \equiv \frac{d!}{i!(d - i)!} \tag{7.79}$$

The second of equations 7.79 defines the binomial coefficient.

The function $P_{i,d}(t)$ is given by equation 7.79 when $0 \le i \le d$; otherwise $P_{i,d}(t)$ is zero. Equations 7.79 can be used to derive the recursion relation for the Bernstein polynomials shown below.

$$P_{i,d}(t) = (1 - t)P_{i,d-1}(t) + tP_{i-1,d-1}(t) \tag{7.80}$$

Equation 7.80 can be proved by substituting equations 7.79 for the Bernstein polynomials and by collecting terms. When $i = 0$ in equation 7.80, the second term on the right is zero because for this case the subscript $i-1$ is less than zero (and thus outside of the subscript range for which $P_{i,d}(t)$ is nonzero). Similarly, when $i = d$ in equation 7.80, the first term on the right is zero because the subscript i is greater than the second subscript $d-1$.

Equations 7.78–7.80 can be programmed to provide the subroutine bezier() listed below. This subroutine plots a general Bézier curve, starting with the coordinates of n control points.

```
void bezier(int n,double x[],double y[])
{
    int i,j,k,d,nsegs;
    double t,dt;
    double p[20][20],xp,yp;

    d = n-1;
    nsegs = 50; dt = 1./(double) nsegs;
    MoveTo((int) x[0],(int) y[0]);

    for (j=1,t=dt;j<=nsegs;j++,t+=dt)
    {
        p[0][0] = 1.;
        for (k=1; k<=d; k++)
        {
            p[0][k] = (1.-t)*p[0][k-1];
            for (i=1; i<k; i++)
            {
                p[i][k] = (1.-t)*p[i][k-1] + t*p[i-1][k-1];
            }
            p[k][k] = t*p[k-1][k-1];
        }
        xp = 0.; yp =0.;
        for (i=0; i<=d; i++)
        {
            xp = xp + x[i]*p[i][d];
            yp = yp + y[i]*p[i][d];
        }
        LineTo((int) xp,(int) yp);
    }
}
```

For the general-order Bézier curve, each vertex point has an effect on every part of the curve. Therefore, there is little local control by an individual vertex point, especially for curves that have a large number of vertices. To achieve local control of the curve shape, cubic Bézier curves may be appropriately joined to form a composite cubic Bézier curve.

Composite Cubic Bézier Curves

PostScript printers use font outlines described by composite cubic Bézier curves. Because a cubic Bézier curve is determined by four vertex points, the overall curve is divided into segments that have four points each. Consider an even total number of data points greater than six. The open curve drawn by the subroutine cubicBezier() listed below is an approximation curve that passes through the first and last points, with the remaining data points acting as control points that influence the curve locally. Additional points are added at the curve joints. Therefore, an interior curve segment has two interior data points and two added vertex points at its ends, whereas an end segment has three data points and one added vertex point.

The curve segments will have matching slopes at the interior joints if the added vertices (joint positions) are colinear with the data points on either side. In the program listing below, C^1 continuity is achieved by placing the added vertex points at the midpoint of the line joining two data points; see figure 7.22a. This program subroutine was used to draw the letter *S* shown in figure 7.22b. In this figure, the original vertex points are shown as dots, and the added vertices are located where the tick marks cross the curves.

```
void cubicBezier(int n,double x[],double y[])
{
    int i,j,nsegs,m,remainder;
    double t,dt,p0,p1,p2,p3,xp,yp,xv[50],yv[50];

    m = 3*n/2 - 3;
    xv[0] = x[0]; yv[0] = y[0];
    j = 1;
    for (i=1; i<m; i++)
```

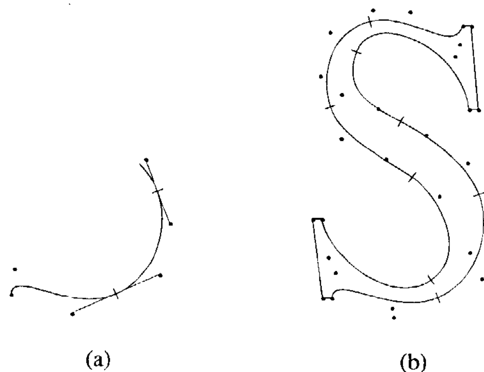


Figure 7.22 Composite cubic Bézier curves: (a) the first two curve segments at the bottom of the letter *S*, starting on the left, with the first joint (shown by a tick mark) located at the midpoint of a line joining the third and fourth data points, and the second joint at the midpoint of a line joining the fifth and sixth data points; (b) the completed letter *S* composed of two composite Bézier curves and six short line segments that form the serifs at the ends of the curves.

```
    {
        remainder = i % 3;
        if (remainder == 0)
        {
            xv[i] = .5*(x[j]+x[j-1]);
            yv[i] = .5*(y[j]+y[j-1]);
        }
        else
        {
            xv[i] = x[j]; yv[i] = y[j];
            j++;
        }
    }
    xv[m] = x[n-1]; yv[m] = y[n-1];

    nsegs = 50; dt = 1./(double) nsegs;
    MoveTo((int) xv[0],(int) yv[0]);
    for (i=0; i<m-2; i+=3)
    {
        for (j=1,t=dt;j<=nsegs;j++,t+=dt)
        {
            p0 = (1.-t)*(1.-t)*(1.-t);
            p1 = 3.*t*(1.-t)*(1.-t);
            p2 = 3.*t*t*(1.-t);
            p3 = t*t*t;
            xp = p0*xv[i]+p1*xv[i+1]+p2*xv[i+2]+p3*xv[i+3];
            yp = p0*yv[i]+p1*yv[i+1]+p2*yv[i+2]+p3*yv[i+3];
            LineTo((int) xp,(int) yp);
        }
    }
}
```

EXERCISES

7.1 Use the Lagrange interpolating polynomial formulation to implement the high-order polynomial curve-fitting method discussed in section 7.1. Apply your program to various polylines, and find cases for which large wiggles occur between data points.

7.2 Generalize the high-order polynomial curve fit of section 7.1 by incorporating a parametric representation for x and y following the discussion in section 7.3. Program your results and carry out a test on an open curve that first extends to larger x values and then turns back to smaller x values.

7.3 On your CAD system or with programming, reproduce the canoe cross sections shown at the beginning of this chapter. The data for these curves are given in table 12.3 in chapter 12.

7.4 On your CAD system, draw the curves representing the ceiling rib structure shown in figure 7.4.

7.5 Program the circular arc curve-fit developed in section 7.2. Extend the subroutine to include the cases of lines with two, three, and four points.

7.6 Write and test a program for a circular arc fit of *closed* curves.

7.7 Write a program to plot the parabolic blend for *closed* curves.

7.8 Use the subroutine given in section 7.5 to write a program for plotting natural cubic splines for open curves.

7.9 Implement the subroutines of section 7.6 to draw both open and closed quadratic uniform B-splines.

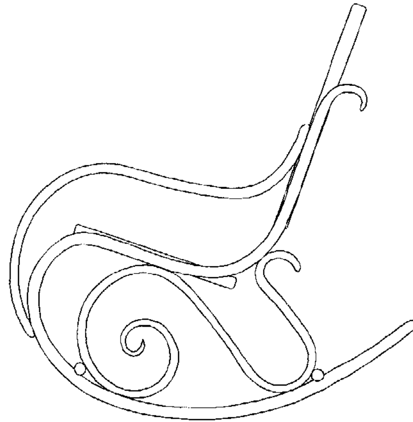
7.10 Implement the subroutine of section 7.6 to draw open cubic uniform B-splines.

7.11 Write a program for drawing *closed* cubic uniform B-splines.

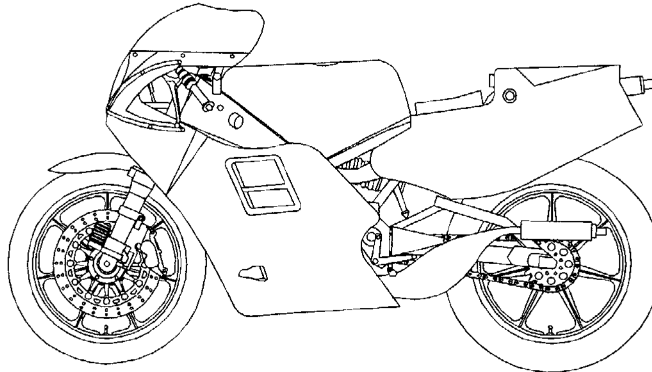
7.12 Use AutoCAD to draw your initials in a particular font style (the Times Roman font was used for the example in figure 7.16). Draw the first letter with a circular-arc fit, the second letter with quadratic B-splines, and the third letter with cubic B-splines.

7.13 Use cubic B-splines to draw the automobile curves shown in figure 7.15.

7.14 Use cubic B-splines to draw the side view (shown below) of the bentwood rocker designed by Michael Thonet in 1860 (see, e.g., C. D. Gandy and S. Zimmermann-Stidham, *Contemporary Classics: Furniture of the Masters.* New York: Whitney Library of Design, 1981).



7.15 Use B-splines and arrays to draw the side view of a vehicle, like the Yamaha motorcycle below, drawn on AutoCAD by Nam Won Back.



7.16 Implement the subroutine in section 7.7 to draw an open quadratic nonuniform B-spline.

7.17 Derive the algebraic expressions for the fourth-order (cubic) nonuniform B-spline basis function $N_{i,4}(t)$.

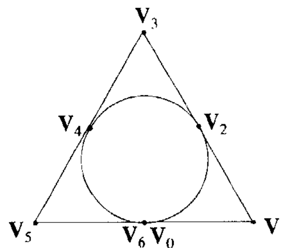
7.18 Write a program that draws fourth-order nonuniform B-splines.

7.19 For uniformly spaced knots, show that the fourth-order basis functions derived in exercise 7.17 reduce to the $b_i(t)$ expressions 7.61, except for the first two and last two segments along the curve. Derive the appropriate expressions for the $b_i(t)$ terms in the first two segments and in the last two segments.

7.20 Write a program to draw quadratic nonuniform rational B-splines from given control points, knot vector, and weight vector.

7.21 Write a quadratic NURBS program to produce the curves shown in figures 7.17b and 7.21.

7.22 Use the program of exercise 7.20 to plot a circle by using the seven control vertices located on the equilateral triangle shown below (the first and last control points lie at the midpoint of the equilateral triangle base). The following are the appropriate knot and weight vectors: $\mathbf{T}_{knot}$ = [0, 0, 0, 1, 1, 2, 2, 3, 3, 3] and $\mathbf{w}$ = [1, 0.5, 1, 0.5, 1, 0.5, 1].



7.23 Implement the composite cubic Bézier curve subroutine of section 7.8, and then plot the curved portions of the letter *S* shown in figure 7.22b.

BIBLIOGRAPHY

ANAND, V. B., *Computer Graphics and Geometric Modeling for Engineers.* New York: John Wiley & Sons, Inc., 1993.

AutoCAD Release 12 Reference Manual. Sausalito, Calif.: Autodesk, Inc., 1992.

AutoSurf Release 2 Reference Manual. Sausalito, Calif.: Autodesk, Inc., 1993.

BARTELS, R. H., J. C. BEATTY, AND B. A. BARSKY, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* Los Altos, Calif.: Morgan Kaufmann Publishers, Inc., 1987.

FARIN, G., *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide,* 2nd ed. San Diego, Calif: Academic Press, Inc., 1990.

PIEGL, L. A., "Rational B-spline Curves and Surfaces for CAD and Graphics," pp. 225–269 in *State of the Art in Computer Graphics: Visualization and Modeling,* D. F. Rogers and R. A. Earnshaw, editors. New York: Springer-Verlag, 1991.

RENNER, G., AND V. POCHOP, "A New Method for Local Smooth Interpolation," pp. 137–147 in *Eurographics '81,* J. L. Encarnacao, editor. Amsterdam: North-Holland Publishing Co., 1981.

ROGERS, D. F., AND J. A. ADAMS, *Mathematical Elements for Computer Graphics,* 2nd ed. New York: McGraw-Hill Publishing Co., 1990.