CHAPTER

FIVE

Fundamental Geometric Algorithms

5.1 Introduction

In this chapter we present five tools which are fundamental in the implementation of B-spline curves and surfaces; these are knot insertion, knot refinement, knot removal, degree elevation, and degree reduction. We devote a section to each topic, and the layout of each section is roughly:

- a statement of the problem (for curves);
- a list of applications;
- clarification of the problem and solution approaches (curves);
- a list of references where more rigorous derivations and proofs can be found;
- the solution formulas (curves);
- worked examples (curves);
- computer algorithms;
- examples of applications;
- the solution for surfaces, and outlines of the surface algorithms.

5.2 Knot Insertion

Let $\mathbf{C}^w(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i^w$ be a NURBS curve defined on $U = \{u_0, \ldots, u_m\}$. Let $\bar{u} \in [u_k, u_{k+1})$, and insert $\bar{u}$ into U to form the new knot vector $\bar{U} = \{\bar{u}_0 = u_0, \ldots, \bar{u}_k = u_k, \bar{u}_{k+1} = \bar{u}, \bar{u}_{k+2} = u_{k+1}, \ldots, \bar{u}_{m+1} = u_m\}$. If $\mathcal{V}_U$ and $\mathcal{V}_{\bar{U}}$ denote the vector spaces of curves defined on U and $\bar{U}$, respectively, then clearly $\mathcal{V}_U \subset \mathcal{V}_{\bar{U}}$ (and $\dim(\mathcal{V}_{\bar{U}}) = \dim(\mathcal{V}_U) + 1$); thus $\mathbf{C}^w(u)$ has a representation on $\bar{U}$ of the form

$$\mathbf{C}^w(u) = \sum_{i=0}^{n+1} \bar{N}_{i,p}(u)\mathbf{Q}_i^w \tag{5.1}$$

where the $\{\bar{N}_{i,p}(u)\}$ are the pth-degree basis functions on $\bar{U}$. The term *knot insertion* generally refers to the process of determining the $\{\mathbf{Q}_i^w\}$ in Eq. (5.1). It is important to note that knot insertion is really just a change of vector space basis; the curve is not changed, either geometrically or parametrically.

Although not immediately obvious, knot insertion is one of the most important of all B-spline algorithms. Some of its uses are:

- evaluating points and derivatives on curves and surfaces;
- subdividing curves and surfaces;
- adding control points in order to increase flexibility in shape control (interactive design).

Now the $\{\mathbf{Q}_i^w\}$ in Eq. (5.1) can be obtained by setting up and solving a system of linear equations. If we set

$$\sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i^w = \sum_{i=0}^{n+1} \bar{N}_{i,p}(u)\mathbf{Q}_i^w \tag{5.2}$$

then by substituting $n+2$ suitable values of u into Eq. (5.2) we obtain a non-singular, banded system of $n+2$ linear equations in the $n+2$ unknowns, $\mathbf{Q}_i^w$. However, there is a more efficient solution. Property P2.2 and $\bar{u} \in [u_k, u_{k+1})$ imply that

$$\sum_{i=k-p}^{k} N_{i,p}(u)\mathbf{P}_i^w = \sum_{i=k-p}^{k+1} \bar{N}_{i,p}(u)\mathbf{Q}_i^w \tag{5.3}$$

for all $u \in [u_k, u_{k+1})$, and

$$\begin{aligned} N_{i,p}(u) &= \bar{N}_{i,p}(u) \qquad i = 0, \ldots, k-p-1 \\ N_{i,p}(u) &= \bar{N}_{i+1,p}(u) \qquad i = k+1, \ldots, n \end{aligned} \tag{5.4}$$

Equations (5.3) and (5.4), together with the linear independence of the basis functions (Section 2.4), imply that

$$\begin{aligned} \mathbf{P}_i^w &= \mathbf{Q}_i^w \qquad i = 0, \ldots, k-p-1 \\ \mathbf{P}_i^w &= \mathbf{Q}_{i+1}^w \qquad i = k+1, \ldots, n \end{aligned} \tag{5.5}$$

Now consider the $N_{i,p}(u)$ for $i = k-p, \ldots, k$. They can be expressed in terms of the $\bar{N}_{i,p}(u)$ when $i = k-p, \ldots, k+1$, by

$$N_{i,p}(u) = \frac{\bar{u} - \bar{u}_i}{\bar{u}_{i+p+1} - \bar{u}_i}\bar{N}_{i,p}(u) + \frac{\bar{u}_{i+p+2} - \bar{u}}{\bar{u}_{i+p+2} - \bar{u}_{i+1}}\bar{N}_{i+1,p}(u) \tag{5.6}$$

Equation (5.6) is proven by induction on p (and using Eq. [2.5]), but we omit the proof here as it is quite messy. Proofs using divided differences are found in [DeBo78; Boeh80; Lee83].

For brevity we now write $\bar{N}_i$ for $\bar{N}_{i,p}(u)$. Substituting Eq. (5.6) into Eq. (5.3) yields

$$\begin{aligned} &\left(\frac{\bar{u} - \bar{u}_{k-p}}{\bar{u}_{k+1} - \bar{u}_{k-p}}\bar{N}_{k-p} + \frac{\bar{u}_{k+2} - \bar{u}}{\bar{u}_{k+2} - \bar{u}_{k-p+1}}\bar{N}_{k-p+1}\right)\mathbf{P}_{k-p}^w \\ &\quad + \left(\frac{\bar{u} - \bar{u}_{k-p+1}}{\bar{u}_{k+2} - \bar{u}_{k-p+1}}\bar{N}_{k-p+1} + \frac{\bar{u}_{k+3} - \bar{u}}{\bar{u}_{k+3} - \bar{u}_{k-p+2}}\bar{N}_{k-p+2}\right)\mathbf{P}_{k-p+1}^w \\ &\quad \vdots \\ &\quad + \left(\frac{\bar{u} - \bar{u}_k}{\bar{u}_{k+p+1} - \bar{u}_k}\bar{N}_k + \frac{\bar{u}_{k+p+2} - \bar{u}}{\bar{u}_{k+p+2} - \bar{u}_{k+1}}\bar{N}_{k+1}\right)\mathbf{P}_k^w \\ &= \bar{N}_{k-p}\mathbf{Q}_{k-p}^w + \cdots + \bar{N}_{k+1}\mathbf{Q}_{k+1}^w \end{aligned}$$

By equating coefficients and using the knot vector U in place of $\bar{U}$, we obtain

$$\begin{aligned} 0 = {} & \bar{N}_{k-p}\left(\mathbf{Q}_{k-p}^w - \mathbf{P}_{k-p}^w\right) \\ & + \bar{N}_{k-p+1}\left(\mathbf{Q}_{k-p+1}^w - \frac{\bar{u} - u_{k-p+1}}{u_{k+1} - u_{k-p+1}}\mathbf{P}_{k-p+1}^w - \frac{u_{k+1} - \bar{u}}{u_{k+1} - u_{k-p+1}}\mathbf{P}_{k-p}^w\right) \\ & \quad \vdots \\ & + \bar{N}_k\left(\mathbf{Q}_k^w - \frac{\bar{u} - u_k}{u_{k+p} - u_k}\mathbf{P}_k^w - \frac{u_{k+p} - \bar{u}}{u_{k+p} - u_k}\mathbf{P}_{k-1}^w\right) + \bar{N}_{k+1}\left(\mathbf{Q}_{k+1}^w - \mathbf{P}_k^w\right) \end{aligned} \tag{5.7}$$

For $i = k-p+1, \ldots, k$, we set

$$\alpha_i = \frac{\bar{u} - u_i}{u_{i+p} - u_i} \tag{5.8}$$

and note that

$$1 - \alpha_i = \frac{u_{i+p} - \bar{u}}{u_{i+p} - u_i} \tag{5.9}$$

Using the linear independence of the basis functions, and substituting Eqs. (5.8) and (5.9) into Eq. (5.7), yields

$$\begin{aligned} \mathbf{Q}_{k-p}^w &= \mathbf{P}_{k-p}^w \\ \mathbf{Q}_i^w &= \alpha_i\mathbf{P}_i^w + (1-\alpha_i)\mathbf{P}_{i-1}^w \qquad k-p+1 \le i \le k \\ \mathbf{Q}_{k+1}^w &= \mathbf{P}_k^w \end{aligned} \tag{5.10}$$

Finally, by combining Eqs. (5.5) and (5.10) we obtain the formula for computing all the new control points, $\mathbf{Q}_i^w$, of Eq. (5.1), that is

$$\mathbf{Q}_i^w = \alpha_i\mathbf{P}_i^w + (1-\alpha_i)\mathbf{P}_{i-1}^w \tag{5.11}$$

where

$$\alpha_i = \begin{cases} 1 & i \le k-p \\ \dfrac{\bar{u}-u_i}{u_{i+p}-u_i} & k-p+1 \le i \le k \\ 0 & i \ge k+1 \end{cases}$$

Equation (5.11) says that only p new control points must be computed. For brevity we use $\mathbf{P}$ instead of $\mathbf{P}^w$ in examples Ex5.1 – Ex5.4.

Examples

Ex5.1 Let $p = 3$ and $U = \{0,0,0,0,1,2,3,4,5,5,5,5\}$. The control points are $\mathbf{P}_0, \ldots, \mathbf{P}_7$. We insert $\bar{u} = 5/2$. Then $\bar{u} \in [u_5, u_6)$ and $k = 5$. Thus, $\mathbf{Q}_0 = \mathbf{P}_0, \ldots, \mathbf{Q}_2 = \mathbf{P}_2$ and $\mathbf{Q}_6 = \mathbf{P}_5, \ldots, \mathbf{Q}_8 = \mathbf{P}_7$. Applying Eq. (5.11) we find that

$$\alpha_3 = \frac{\frac{5}{2}-0}{3-0} = \frac{5}{6} \implies \mathbf{Q}_3 = \frac{5}{6}\mathbf{P}_3 + \frac{1}{6}\mathbf{P}_2$$

$$\alpha_4 = \frac{\frac{5}{2}-1}{4-1} = \frac{1}{2} \implies \mathbf{Q}_4 = \frac{1}{2}\mathbf{P}_4 + \frac{1}{2}\mathbf{P}_3$$

$$\alpha_5 = \frac{\frac{5}{2}-2}{5-2} = \frac{1}{6} \implies \mathbf{Q}_5 = \frac{1}{6}\mathbf{P}_5 + \frac{5}{6}\mathbf{P}_4$$

Figure 5.1a shows the control polygon before and after the insertion, and Figure 5.1b shows the basis functions before and after the insertion. The bottom part of Figure 5.1a shows the ratios used to subdivide the polygon legs.

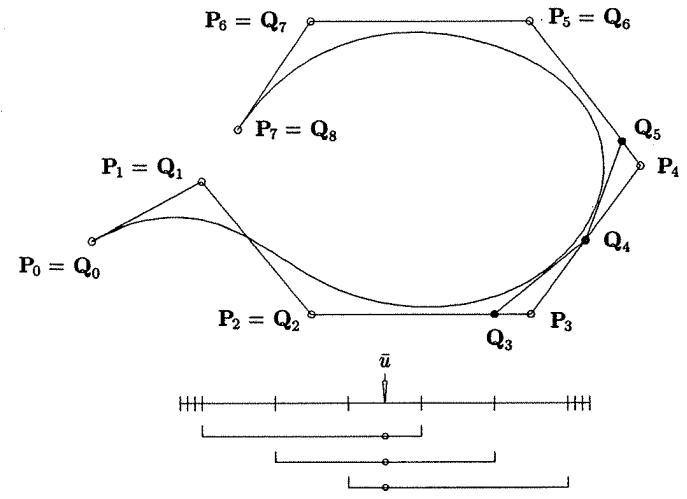
Ex5.2 Use the same curve as in Ex5.1, that is, $p = 3$, $U = \{0,0,0,0,1,2,3,4,5,5,5,5\}$, and $\mathbf{P}_0, \ldots, \mathbf{P}_7$. This time we insert a knot which is already in the knot vector, namely $\bar{u} = 2$. Then $\bar{u} \in [u_5, u_6)$, $k = 5$, and again we compute $\mathbf{Q}_3$, $\mathbf{Q}_4$, and $\mathbf{Q}_5$

$$\alpha_3 = \frac{2-0}{3-0} = \frac{2}{3} \implies \mathbf{Q}_3 = \frac{2}{3}\mathbf{P}_3 + \frac{1}{3}\mathbf{P}_2$$

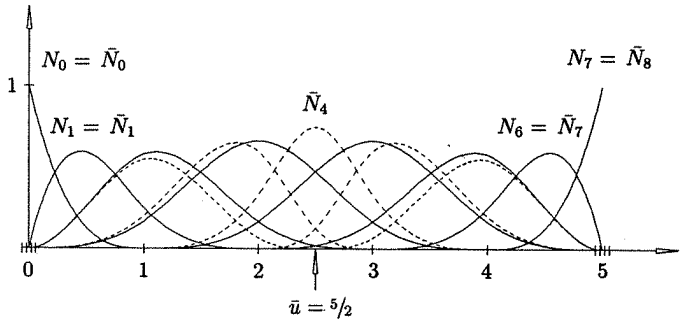
$$\alpha_4 = \frac{2-1}{4-1} = \frac{1}{3} \implies \mathbf{Q}_4 = \frac{1}{3}\mathbf{P}_4 + \frac{2}{3}\mathbf{P}_3$$

$$\alpha_5 = \frac{2-2}{5-2} = 0 \implies \mathbf{Q}_5 = \mathbf{P}_4$$

The control polygons and basis functions, before and after knot insertion, are shown in Figures 5.2a and 5.2b. From Eq. (5.6) one can verify



(a)

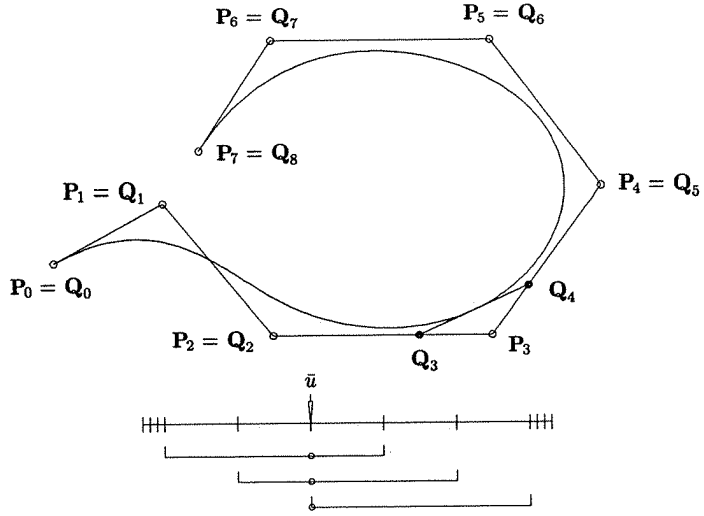


(b)

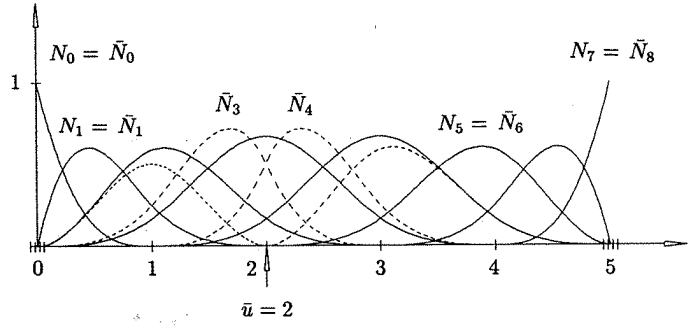
Figure 5.1. Knot insertion into a cubic curve. (a) The control polygon after inserting $u = 5/2$ into the knot vector $\{0,0,0,0,1,2,3,4,5,5,5,5\}$; (b) the original (solid) and the new (dashed) basis functions before and after knot insertion.

that $N_5 = \bar{N}_6$. As we did previously, observe the subdivision of the legs obtained by mapping the appropriate scales shown in the bottom of Figure 5.2a.

Clearly, knot insertion is similar to the deCasteljau algorithm for Bézier curves, and in fact they are equivalent for a curve with no interior knots, i.e., a Bézier curve. However, a glance at Example Ex5.1 and Figure 5.1a shows that the linear interpolation is not the same on each of the p legs of the control polygon. Indeed, in the example, $\alpha_3 = 5/6$, $\alpha_4 = 1/2$, and $\alpha_5 = 1/6$. Based on the previous



(a)



(b)

Figure 5.2. Inserting an existing knot into a cubic curve. (a) The control polygon after inserting $u = 2$ into $\{0,0,0,0,1,2,3,4,5,5,5,5\}$; (b) the original (solid) and the new (dashed) basis functions before and after knot insertion.

examples, we arrive at the general result: For $i = 1,\ldots,n$ let L_i denote the ith-leg of the control polygon

$$L_i = \mathbf{P}^w_{i-1}\mathbf{P}^w_i \qquad \text{and} \qquad d_i = \text{length}\,(L_i) = \mid \mathbf{P}^w_i - \mathbf{P}^w_{i-1} \mid$$

We emphasize that d_i is measured in four-dimensional (homogeneous) space, i.e.

$$d_i = \sqrt{(w_i x_i)^2 + (w_i y_i)^2 + (w_i z_i)^2 + (w_i)^2}$$

With each leg L_i, we associate the knots u_{i+j}, $j = 0,\ldots,p$. Refer to Figures 5.3a and 5.3b and Figures 5.1a and 5.2a. For fixed i let λ_i^j, $j = 0,\ldots,p-1$, denote the length of the knot span $[u_{i+j}, u_{i+j+1})$ relative to $[u_i, u_{i+p})$

$$\lambda_i^j = \frac{u_{i+j+1} - u_{i+j}}{u_{i+p} - u_i} \qquad j = 0,\ldots,p-1 \tag{5.12}$$

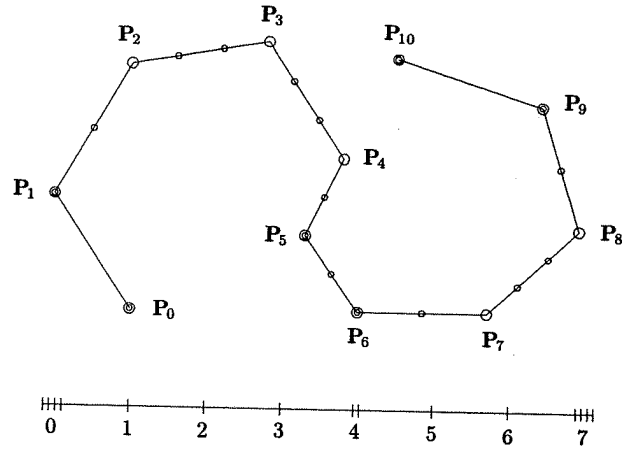
Then L_i is partitioned into segments L_i^j, whose lengths are

$$d_i^j = \lambda_i^j d_i \tag{5.13}$$

respectively. Notice that λ_i^j, and hence d_i^j, can be zero. For example, $d_1^0 = d_1^1 = 0$ and $d_1^2 = d_1$ in Example **Ex5.1**; thus, there is only one segment on the first leg.

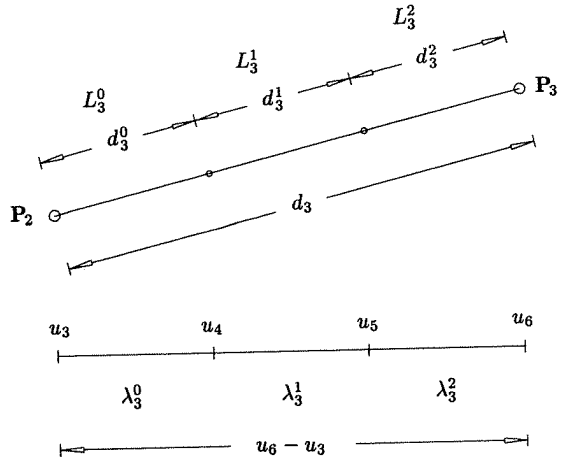
Now Figures 5.3a and 5.3b and Eq. (5.13) show how the polygon legs are subdivided, and Figures 5.1a and 5.2a show how polygon corners are cut in the knot insertion process. In particular, if $\bar{u} \in [u_k, u_{k+1})$ is a knot of multiplicity s, $0 \le s \le p-1$, then the corners with control points $\mathbf{P}_{k-p+1},\ldots,\mathbf{P}_{k-s-1}$ are cut, and the following $p-s$ new control points are generated on the indicated segments

$$\mathbf{Q}^w_{k-j} \in L^j_{k-j} \qquad j = p-1,\ldots,s \tag{5.14}$$



(a)

Figure 5.3. The control polygon and its partitioning by its knot vector. (a) Partitioning of the entire polygon by the knot vector $\{0,0,0,0,1,2,3,4,4,5,6,7,7,7,7\}$; (b) partitioning of the polygon side $\mathbf{P}_2\mathbf{P}_3$ by the knot span $[u_3, u_6]$.



(b)

Figure 5.3. (*Continued.*)

Examples

Ex5.3 Let us check Eqs. (5.12)–(5.14) in Example Ex5.1. We have $k = 5$ and $s = 0$. By Eq. (5.14)

$$\mathbf{Q}_3 \in L_3^2 \qquad \mathbf{Q}_4 \in L_4^1 \qquad \mathbf{Q}_5 \in L_5^0$$

Furthermore

$$\lambda_3^2 = \frac{u_6 - u_5}{u_6 - u_3} = \frac{3-2}{3-0} = \frac{1}{3}$$

$$\lambda_4^1 = \frac{u_6 - u_5}{u_7 - u_4} = \frac{3-2}{4-1} = \frac{1}{3}$$

$$\lambda_5^0 = \frac{u_6 - u_5}{u_8 - u_5} = \frac{3-2}{5-2} = \frac{1}{3}$$

(This is not surprising, since the knots are equally spaced). Recalling that $\alpha_3 = 5/6$, $\alpha_4 = 1/2$, and $\alpha_5 = 1/6$, we see that $\mathbf{Q}_3$, $\mathbf{Q}_4$, and $\mathbf{Q}_5$ are located at the midpoints of their segments, L_3^2, L_4^1, and L_5^0. This follows from the fact that $\bar{u} = 5/2$ is halfway between $u_5 = 2$ and $u_6 = 3$.

Ex5.4 For Example Ex5.2, $k = 5$ and $s = 1$. By Eq. (5.14)

$$\mathbf{Q}_3 \in L_3^2 \qquad \mathbf{Q}_4 \in L_4^1$$

We still have $\lambda_3^2 = \lambda_4^1 = 1/3$. Now $\alpha_3 = 2/3$ and $\alpha_4 = 1/3$, which imply that $\mathbf{Q}_3$ and $\mathbf{Q}_4$ are located at the starting points of their segments, L_3^2 and L_4^1, respectively. This follows from the fact that $\bar{u} = 2$ lies at the start of the knot span, $[u_5, u_6)$.

Figure 5.4 shows the partitioning of the same control polygon as in Figure 5.3a but with the nonuniform knot vector $U = \{0,0,0,0,1,1.5,2.3,3.9,4.3,5,6.5,7,7,7,7\}$.

As we shall see later, it is often necessary to insert a knot multiple times. Equation (5.11) can be generalized to handle this. Suppose $\bar{u} \in [u_k, u_{k+1})$ initially has multiplicity s, and suppose it is to be inserted r times, where $r+s \le p$ (it generally makes no practical sense to have interior knot multiplicities greater than p). Denote the ith new control point in the rth insertion step by $\mathbf{Q}_{i,r}^w$ (with $\mathbf{Q}_{i,0}^w = \mathbf{P}_i^w$). Then $\mathbf{Q}_{i,r}^w$ is

$$\mathbf{Q}_{i,r}^w = \alpha_{i,r}\mathbf{Q}_{i,r-1}^w + (1-\alpha_{i,r})\mathbf{Q}_{i-1,r-1}^w \tag{5.15}$$

where

$$\alpha_{i,r} = \begin{cases} 1 & i \le k-p+r-1 \\ \dfrac{\bar{u}-u_i}{u_{i+p-r+1}-u_i} & k-p+r \le i \le k-s \\ 0 & i \ge k-s+1 \end{cases}$$

If $s = 0$ and $r = p$, then Eq. (5.15) generates a triangular table of control points (see Table 5.1, and see Figure 5.5 with $p = 3$). The outer control points

$$\mathbf{Q}_{k-p+1,1}^w, \mathbf{Q}_{k-p+2,2}^w, \cdots, \mathbf{Q}_{k,p}^w, \cdots, \mathbf{Q}_{k,2}^w, \mathbf{Q}_{k,1}^w$$

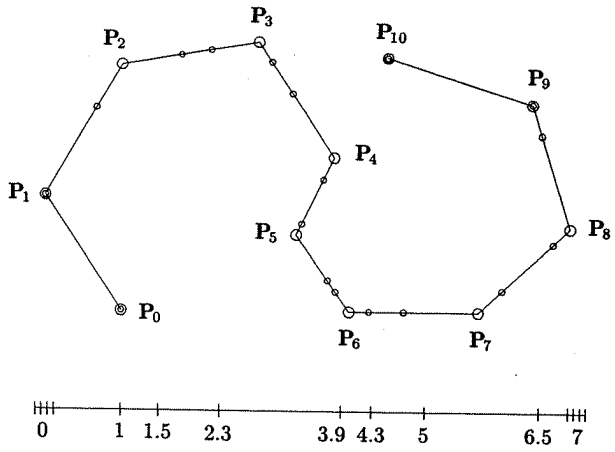


Figure 5.4. Nonuniform partitioning of the control polygon in Figure 5.3a defined by the knot vector $\{0,0,0,0,1,1.5,2.3,3.9,4.3,5,6.5,7,7,7,7\}$.

Table 5.1. Control points generated by Eq. (5.15).

$\mathbf{Q}^w_{k-p+1,1}$			
	$\mathbf{Q}^w_{k-p+2,2}$		
$\mathbf{Q}^w_{k-p+2,1}$			
$\vdots$	$\vdots$	$\cdots$	$\mathbf{Q}^w_{k,p}$
$\mathbf{Q}^w_{k-1,1}$			
	$\mathbf{Q}^w_{k,2}$		
$\mathbf{Q}^w_{k,1}$			

are kept, and the inner ones are discarded. In general, the table is less than full if $s > 0$ and/or $r + s < p$. But in any case, the final control points are obtained by traversing the resulting table in a clockwise fashion, starting at the top of the first column generated. The number of new control points in the last column is $p-(s+r)+1$. In all other columns, two new points are kept (top and bottom); thus, the total number of new (keeper) control points is

$$p - s + r - 1 \tag{5.16}$$

These new control points replace

$$p - s - 1 \tag{5.17}$$

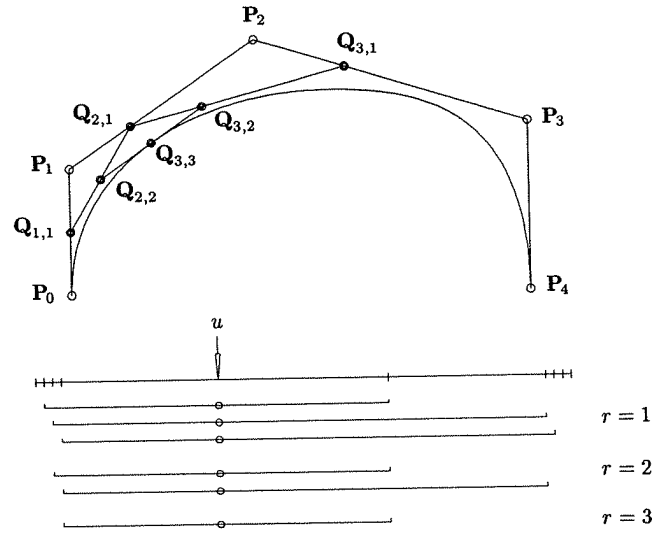


Figure 5.5. Knot insertion into a cubic curve three times. The curve is defined on $\{0,0,0,0,2,3,3,3,3\}$.

old ones, starting at the index

$$k - p + 1 \tag{5.18}$$

Algorithm A5.1 implements Eq. (5.15). It computes the new curve corresponding to the insertion of $\bar{u}$ into $[u_k, u_{k+1})$ r times, where it is assumed that $r + s \leq p$. Rw[] is a local array of length $p + 1$. UP[] and UQ[] are the knot vectors before and after knot insertion, respectively.

```
ALGORITHM A5.1
  CurveKnotIns(np,p,UP,Pw,u,k,s,r,nq,UQ,Qw)
    { /*  Compute new curve from knot insertion  */
      /*  Input:  np,p,UP,Pw,u,k,s,r  */
      /*  Output: nq,UQ,Qw  */
    mp = np+p+1;
    nq = np+r;
      /*  Load new knot vector */
    for(i=0; i<=k; i++)  UQ[i] = UP[i];
    for(i=1; i<=r; i++)  UQ[k+i] = u;
    for(i=k+1; i<=mp; i++)  UQ[i+r] = UP[i];
      /*  Save unaltered control points */
    for(i=0; i<=k-p; i++)  Qw[i] = Pw[i];
    for(i=k-s; i<=np; i++)  Qw[i+r] = Pw[i];
    for(i=0; i<=p-s; i++)  Rw[i] = Pw[k-p+i];
    for(j=1; j<=r; j++)   /* Insert the knot r times */
      {
      L = k-p+j;
      for(i=0; i<=p-j-s; i++)
        {
        alpha = (u-UP[L+i])/(UP[i+k+1]-UP[L+i]);
        Rw[i] = alpha*Rw[i+1] + (1.0-alpha)*Rw[i];
        }
      Qw[L] = Rw[0];
      Qw[k+r-j-s] = Rw[p-j-s];
      }
    for(i=L+1; i<k-s; i++)  /* Load remaining control points */
      Qw[i] = Rw[i-L];
    }
```

We now consider three applications of knot insertion, curve splitting (or subdivision), the evaluation of points and derivatives, and insertion in order to obtain an additional control point for interactive shape design. Refer to Figure 5.5. The original curve, $\mathbf{C}(u)$, is a cubic defined by $U = \{0,0,0,0,2,3,3,3,3\}$ and $\mathbf{P}_0, \ldots, \mathbf{P}_4$. $\bar{u} = 1$ is now inserted three times. Figures 5.6a and 5.6b show the basis functions before and after insertion. The process *splits* the curve. The sets of control points $\{\mathbf{P}_0, \mathbf{Q}_{1,1}, \mathbf{Q}_{2,2}, \mathbf{Q}_{3,3}\}$ and $\{\mathbf{Q}_{3,3}, \mathbf{Q}_{3,2}, \mathbf{Q}_{3,1}, \mathbf{P}_3, \mathbf{P}_4\}$ define cubic B-spline curves, $\mathbf{C}_l(u)$ and $\mathbf{C}_r(u)$, on the knot vectors $U_l = \{0,0,0,0,1,1,$

1,1} and $U_r = \{1,1,1,1,2,3,3,3,3\}$, respectively. Furthermore, $\mathbf{C}_l(u)$ is $\mathbf{C}(u)$ restricted to the domain $u \in [0,1]$, and $\mathbf{C}_r(u)$ is $\mathbf{C}(u)$ restricted to $u \in [1,3]$. Curve splitting, together with the convex hull property (P4.10), forms the basis for "divide and conquer" algorithms using NURBS. Figure 5.7 shows an example of recursive curve splitting: The original curve $\mathbf{C}$ is split at $u = 1/2$, resulting in $\mathbf{C}_1$ and $\mathbf{C}_2$, which are then split at $1/8$ and $3/8$, respectively, yielding $\mathbf{C}_{1,1}$ and $\mathbf{C}_{1,2}$, and $\mathbf{C}_{2,1}$ and $\mathbf{C}_{2,2}$.

Figures 5.5 and 5.6b also show that knot insertion can be used to evaluate points and derivatives on curves. Clearly, $\mathbf{C}(1) = \mathbf{Q}_{3,3}$; furthermore, the derivatives from *both the left and right* can easily be computed using the starting point and endpoint derivative formulas, Eqs. (3.7), (3.9), (3.10), (4.9), and (4.10).

Algorithm A5.2 computes a point on a curve using knot insertion ("corner cutting"). It assumes a routine, FindSpanMult, which finds the knot span, k, and the multiplicity, s.

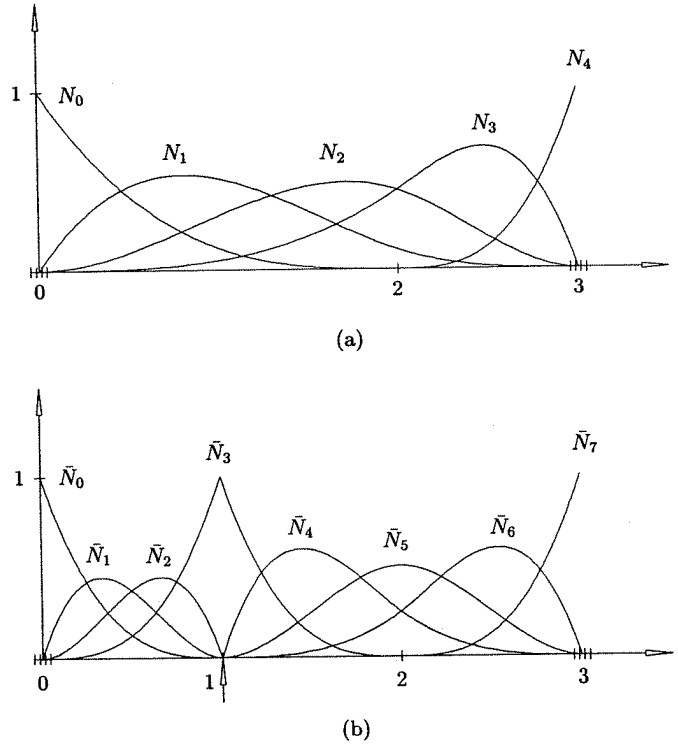


Figure 5.6. Basis functions corresponding to Figure 5.5. (a) Before knot insertion; (b) after knot insertion.

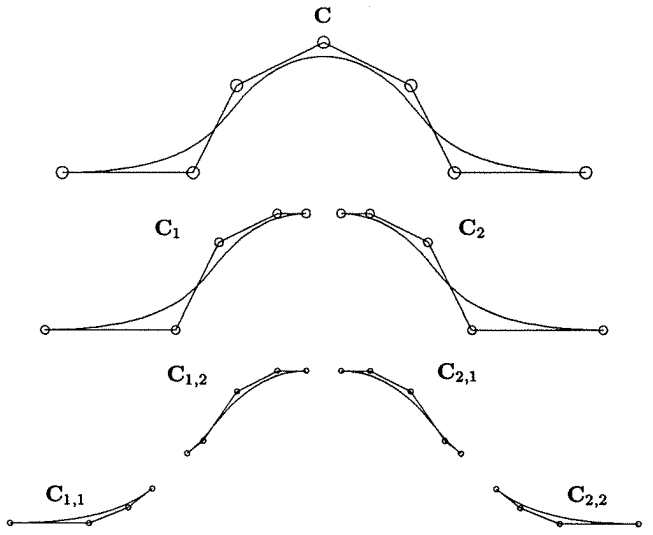


Figure 5.7. Splitting a cubic curve defined on $\{0,0,0,0,1/4,1/2,3/4,1,1,1,1\}$.

```
ALGORITHM A5.2
  CurvePntByCornerCut(n,p,U,Pw,u,C)
    { /*  Compute point on rational B-spline curve  */
      /*  Input:  n,p,U,Pw,u  */
      /*  Output: C  */
    if (u == U[0])  /* Endpoints are special cases */
      {
      C = Pw[0]/w;   return;
      }
    if (u == U[n+p+1])
      {
      C = Pw[n]/w;   return;
      }
    FindSpanMult(n,p,u,U,&k,&s);   /* General case */
    r = p-s;
    for (i=0; i<=r; i++)   Rw[i] = Pw[k-p+i];
    for (j=1; j<=r; j++)
      for (i=0; i<=r-j; i++)
        {
        alfa = (u-U[k-p+j+i])/(U[i+k+1]-U[k-p+j+i]);
        Rw[i] = alfa*Rw[i+1] + (1.0-alfa)*Rw[i];
```

```
    }
C = Rw[0]/w;
}
```

The third application is interactive shape design. As we saw in Chapter 4, curves and surfaces can be shaped by moving control points and/or modifying weights. For modification purposes, a designer might like to have a control point somewhere on the control polygon where one currently does not exist. For example, suppose he picks the point $\mathbf{Q}$ on the three-dimensional polygon leg, $\mathbf{P}_{i-1}\mathbf{P}_i$ (see Figure 5.8). Denote the corresponding four-dimensional point on the leg, $L_i = \mathbf{P}^w_{i-1}\mathbf{P}^w_i$, by $\mathbf{Q}^w$. Then there exists a $\bar{u} \in [u_i, u_{i+p})$ such that a single insertion of $\bar{u}$ causes $\mathbf{Q}^w$ to become a new control point and $\mathbf{Q}$ its projection. This process of determining $\bar{u}$ is called *inverse knot insertion* [Pieg89c]. Now there exists a value s, $0 \le s \le 1$, such that

$$\mathbf{Q}^w = (1-s)\mathbf{P}^w_{i-1} + s\mathbf{P}^w_i$$

which upon projection yields

$$\mathbf{Q} = \frac{(1-s)w_{i-1}\mathbf{P}_{i-1} + sw_i\mathbf{P}_i}{(1-s)w_{i-1} + sw_i} \tag{5.19}$$

It follows that

$$s = \frac{w_{i-1} \mid \mathbf{Q} - \mathbf{P}_{i-1} \mid}{w_{i-1} \mid \mathbf{Q} - \mathbf{P}_{i-1} \mid + w_i \mid \mathbf{P}_i - \mathbf{Q} \mid} \tag{5.20}$$

and thus

$$\bar{u} = u_i + s(u_{i+p} - u_i) \tag{5.21}$$

Knots are inserted into surfaces by simply applying the previous formulas and algorithms to the rows and/or columns of control points. In particular, let $\mathbf{P}^w_{i,j}$,

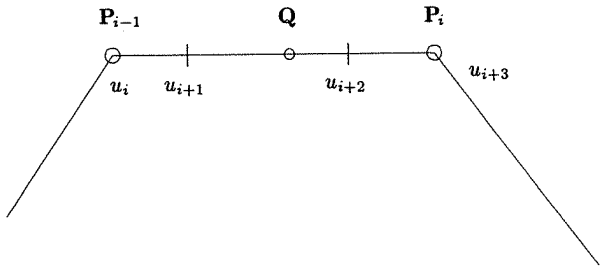


Figure 5.8. Inverse knot insertion for cubic curves; the point $\mathbf{Q}$ lies on the leg $\mathbf{P}_{i-1}\mathbf{P}_i$ partitioned by the knots u_i, u_{i+1}, u_{i+2}, and u_{i+3}.

$0 \le i \le n$, $0 \le j \le m$, be the control points of a NURBS surface; call i the row index and j the column index. Then $\bar{u}$ is added to the knot vector U by doing a $\bar{u}$ knot insertion on each of the $m+1$ columns of control points. The resulting control net is $\mathbf{Q}^w_{i,j}$, $0 \le i \le n+1$, $0 \le j \le m$. Analogously, $\bar{v}$ must be inserted on each of the $n+1$ rows of control points. The resulting control net is $\mathbf{Q}^w_{i,j}$, $0 \le i \le n$, $0 \le j \le m+1$. Figure 5.9 shows a cubic × quadratic surface on $U = \{0,0,0,0,1,1,1,1\}$ and $V = \{0,0,0,1/2,1,1,1\}$. Figures 5.10a and 5.10b show insertion of the knots $\bar{u} = 4/10$ and $\bar{v} = 7/10$, respectively, and Figure 5.10c shows the insertion of both knots. We point out that a surface knot insertion algorithm should not merely consist of a loop in which Algorithm `A5.1` is called $m+1$ (or $n+1$) times. The computation of the alphas (`alpha` in `A5.1`) does not depend on the control points. Hence, they should be computed only once and stored in a local array before entering the loop which executes the $m+1$ or $n+1$ knot insertions. Algorithm `A5.3` is such an algorithm.

```
ALGORITHM A5.3
  SurfaceKnotIns(np,p,UP,mp,q,VP,Pw,dir,uv,k,s,r,nq,UQ,mq,VQ,Qw)
    { /* Surface knot insertion  */
      /* Input:  np,p,UP,mp,q,VP,Pw,dir,uv,k,s,r  */
      /* Output: nq,UQ,mq,VQ,Qw  */
    if (dir == U_DIRECTION)
```

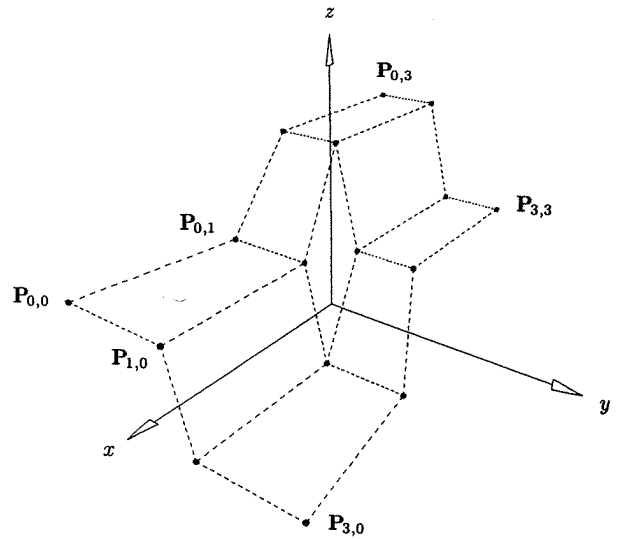
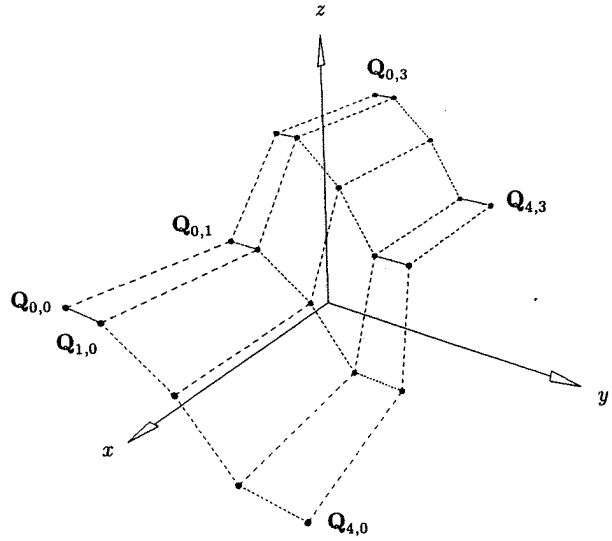
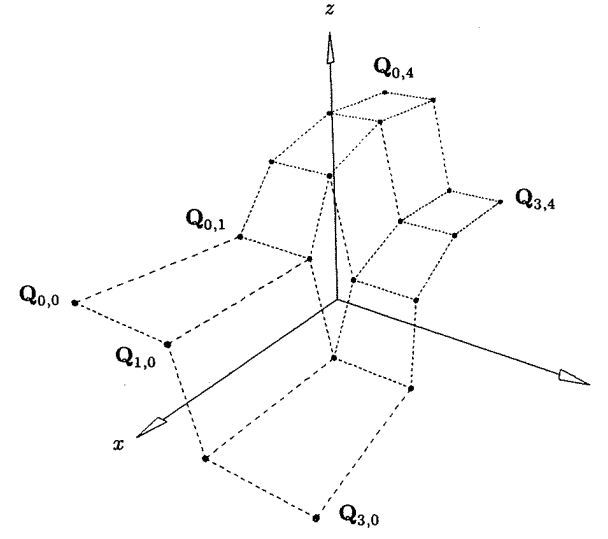


Figure 5.9. The control net of a (cubic × quadratic) surface defined on the knot vectors $U = \{0,0,0,0,1,1,1,1\}$ and $V = \{0,0,0,1/2,1,1,1\}$.

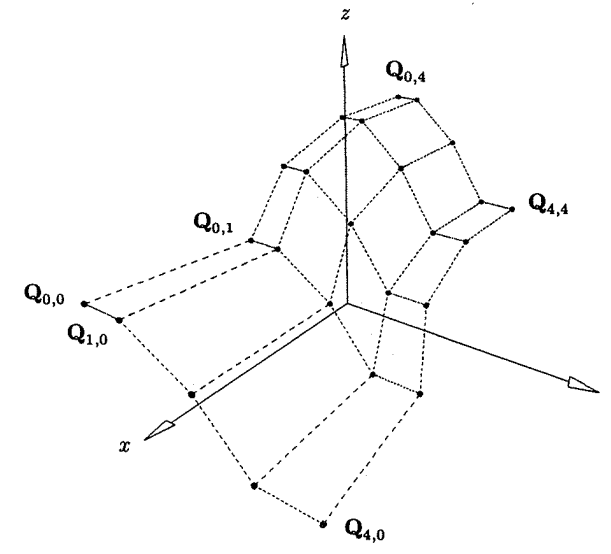


(a)



(b)

Figure 5.10. Knot insertion into the surface in Figure 5.9. (a) Inserting $u = 2/5$ one time in the u direction; (b) inserting $v = 7/10$ one time in the v direction; (c) inserting both knots at the same time.



(c)

Figure 5.10. (*Continued.*)

```
{
load u-vector as in A5.1
copy v-vector into VQ
    /* Save the alphas */
for (j=1; j<=r; j++)
  {
  L = k-p+j;
  for (i=0; i<=p-j-s; i++)
    alpha[i][j] = (uv-UP[L+i])/(UP[i+k+1]-UP[L+i]);
  }
for (row=0; row<=mp; row++)   /* For each row do */
  {
        /* Save unaltered control points */
  for (i=0; i<=k-p; i++)   Qw[i][row] = Pw[i][row];
  for (i=k-s; i<=np; i++)   Qw[i+r][row] = Pw[i][row];
        /* Load auxiliary control points.  */
  for (i=0; i<=p-s; i++)   Rw[i] = Pw[k-p+i][row];
  for (j=1; j<=r; j++)   /* Insert the knot r times */
    {
    L = k-p+j;
    for (i=0; i<=p-j-s; i++)
```

```
        Rw[i] = alpha[i][j]*Rw[i+1] + (1.0-alpha[i][j])*Rw[i];
      Qw[L][row] = Rw[0];
      Qw[k+r-j-s][row] = Rw[p-j-s];
      }
          /* Load the remaining control points */
    for (i=L+1; i<k-s; i++)   Qw[i][row] = Rw[i-L];
    }
  }
if (dir == V_DIRECTION)
  {
  Similar code as above with u- and v-directional
    parameters switched.
  }
}
```

We conclude this section with remarks on surface knot insertion.

- Surface subdivision: Let $\bar{u}$ and $\bar{v}$ be interior parameter values. Inserting$\bar{u}$ p times and $\bar{v}$ q times subdivides the surface into four B-spline (sub-)surfaces. This fact, together with the convex hull property, implies that recursive subdivision is applicable to B-spline surface problems. Figure 5.11 shows a surface to be subdivided. In Figures 5.12–5.14, subdivisions in the u- and v- and in both directions, respectively, are illustrated;

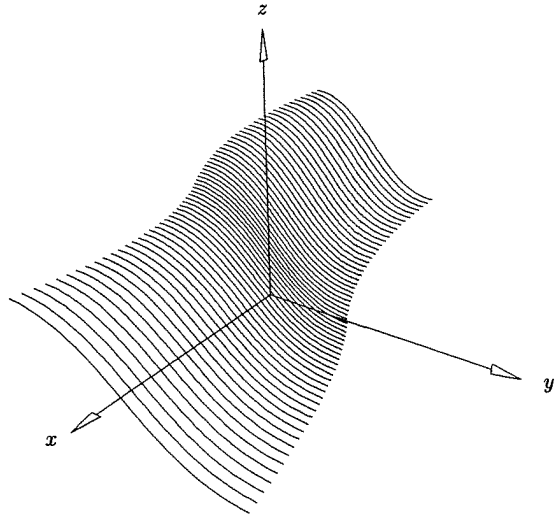
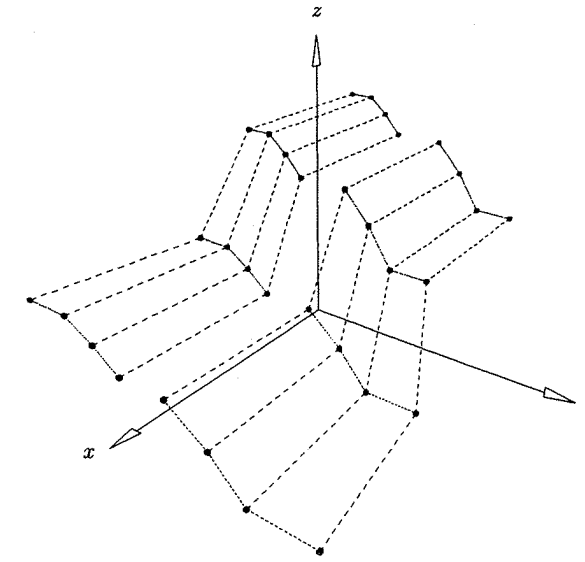
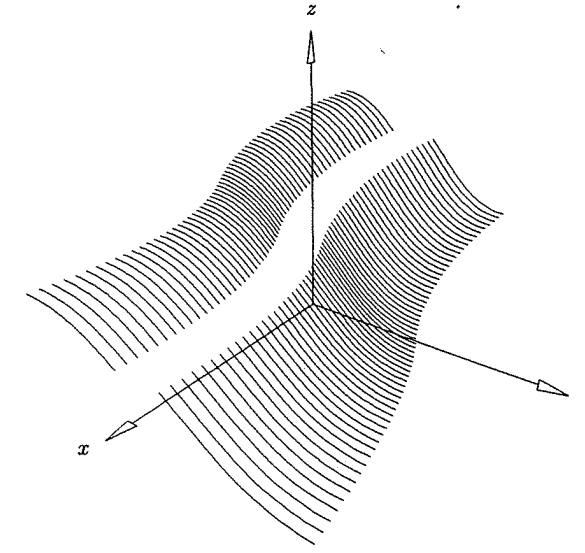


Figure 5.11. A (cubic × quadratic) surface defined as in Figure 5.9.

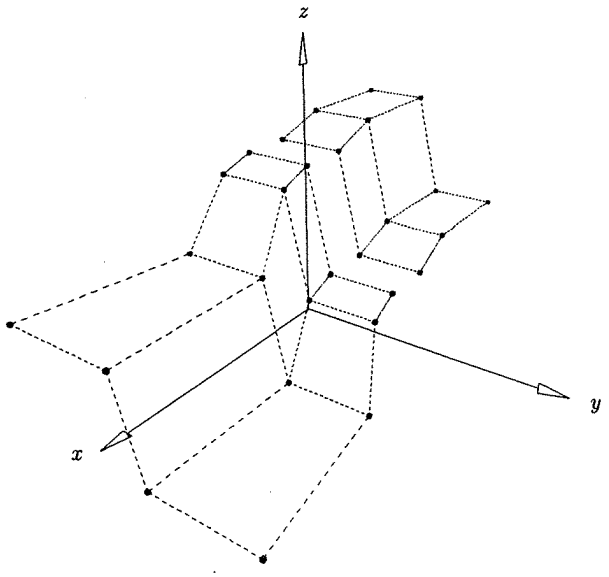


(a)

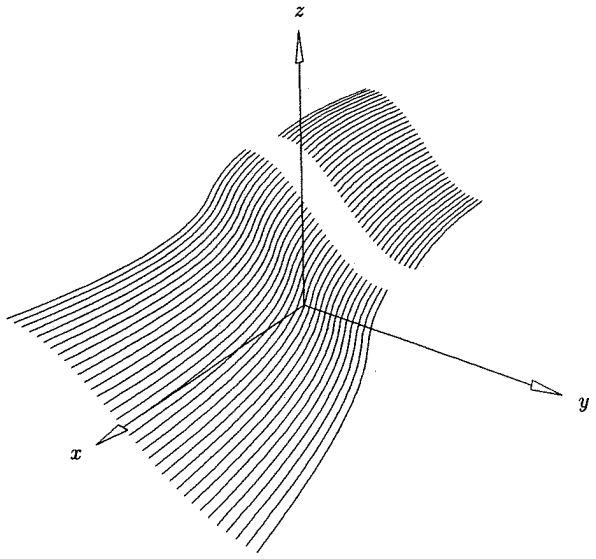


(b)

Figure 5.12. Splitting the surface in Figure 5.11 in the v direction at $u = 2/5$. (a) The control net of the split surface; (b) the split surface.

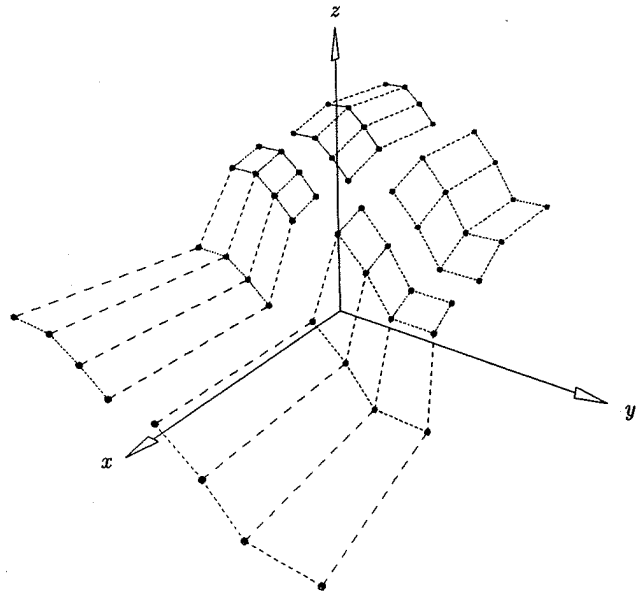


(a)



(b)

Figure 5.13. Splitting the surface in Figure 5.11 in the u direction at $v = 7/10$. (a) The control net of the split surface; (b) the split surface.

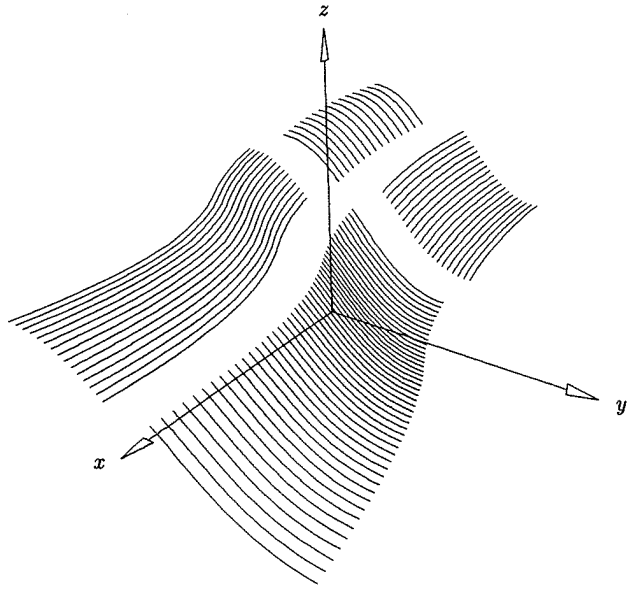


(a)

Figure 5.14. Splitting the surface in Figure 5.11 in both u and v directions at $u = 2/5$ and $v = 7/10$. (a) The control net of the split surface; (b) the split surface.

- Curve extraction: To extract the isoparametric curve, $\mathbf{C}_{\bar{u}}^{w}(v)$, we must do p $\bar{u}$ knot insertions on each of the $m+1$ columns of control points, i.e., a total of $p(m+1)$ knot insertions; this yields the $\{\mathbf{Q}_j^w(\bar{u})\}$ of Eq. (4.16). Extracting a curve, $\mathbf{C}_{\bar{v}}^{w}(u)$, requires $q(n+1)$ knot insertions; Figure 5.15 shows an example of this;
- Surface evaluation: Inserting $\bar{u}$ p times and $\bar{v}$ q times causes the surface point, $\mathbf{S}(\bar{u},\bar{v})$, to become a corner control point of the resulting four subsurfaces. A corner cutting algorithm similar to Algorithm `A5.2` can be developed. Either $\bar{u}$ or $\bar{v}$ may be inserted first, but as was the case for Bézier surface evaluation using the deCasteljau algorithm, the computational complexity is order-dependent if $p \neq q$ (see Eqs. [1.25] and [1.26]). Hence, we can compute $\mathbf{S}(\bar{u},\bar{v})$ with either
  - $p(q+1)$ $\bar{u}$ knot insertions, plus q $\bar{v}$ knot insertions;
  - $q(p+1)$ $\bar{v}$ knot insertions, plus p $\bar{u}$ knot insertions.

  Using corner point derivative formulas (for example, Eqs. [3.24], [4.24], [4.25], and [4.26]), the partial derivatives and normal vectors from all four directions (left and right for both u and v) are obtained.



(b)

Figure 5.14. (*Continued.*)

5.3 Knot Refinement

Knot insertion concerned itself with inserting a single knot, possibly multiple times. It is often necessary to insert many knots at once; this is called *knot refinement.* To state the problem, let $\mathbf{C}^w(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i^w$ be defined on the knot vector $U = \{u_0, \ldots, u_m\}$, and let $X = \{x_0, \ldots, x_r\}$ satisfy $x_i \leq x_{i+1}$ and $u_0 < x_i < u_m$ for all i. The elements of X are to be inserted into U, and the corresponding new set of control points, $\{\mathbf{Q}_i^w\}$, $i = 0, \ldots, n+r+1$, is to be computed. New knots, x_i, should be repeated in X with their multiplicities; e.g., if x and y $(x < y)$ are to be inserted with multiplicities 2 and 3, respectively, then $X = \{x, x, y, y, y\}$. Clearly, knot refinement can be accomplished by multiple applications of knot insertion. However, we distinguish the two problems here because there exist more efficient algorithms for knot refinement (see [Cohe80; Boeh85a, 85b; Lych85]).

The applications of knot refinement include:

- decomposition of B-spline curves and surfaces into their constituent (Bézier) polynomial pieces – we elaborate on this later;
- merging of two or more knot vectors in order to obtain a set of curves which are defined on one common knot vector; as we see in subsequent chapters, this is important in constructing certain types of surfaces;

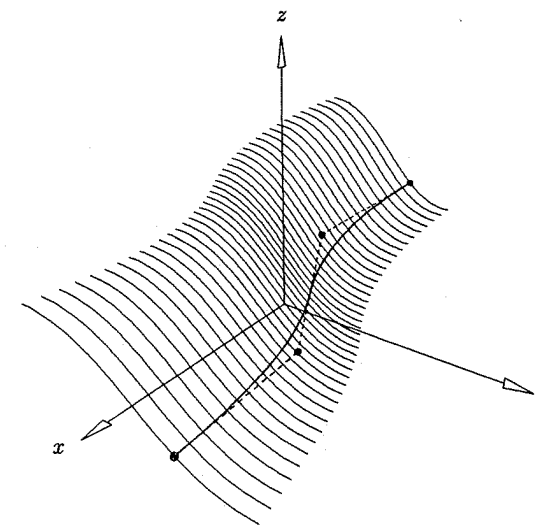


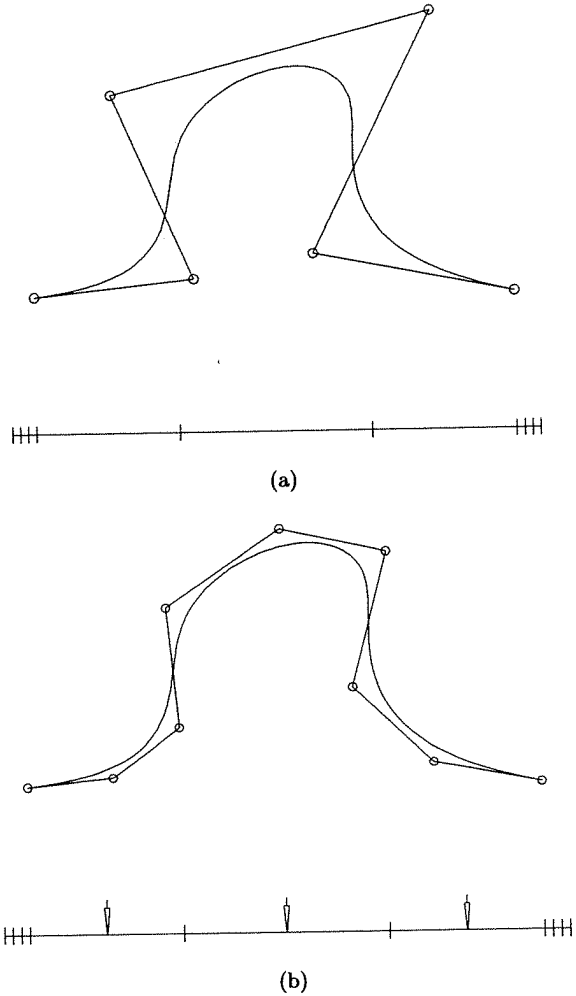
Figure 5.15. Extracting a surface isoparametric curve via knot insertion.

- obtaining polygonal (polyhedral) approximations to curves (surfaces); refining knot vectors brings the control polygon (net) closer to the curve (surface), and in the limit the polygon (net) converges to the curve (surface). A proof of this is found in [Lane80; DeBo87]. Figures 5.16b and 5.16c show one and two midpoint knot refinements applied to the cubic curve of Figure 5.16a (a new knot is inserted at the midpoint of each knot span). Figures 5.17a to 5.17c show three midpoint knot refinements to the surface of Figure 5.9, in the u-, v-, and in both directions, respectively.

We require no additional mathematics to develop an algorithm for knot refinement; it is really just a software problem. The solution steps are:

1. find indices a and b such that $u_a \leq x_i < u_b$ for all i;
2. from Eqs. (5.14) and (5.18) it follows that the control points $\mathbf{P}_0^w, \ldots, \mathbf{P}_{a-p}^w$ and $\mathbf{P}_{b-1}^w \ldots, \mathbf{P}_n^w$ do not change; therefore, copy these to the appropriate $\mathbf{Q}_i^w$ locations, leaving room for the $r+p+b-a-1$ new control points;
3. denote the new knot vector by $\bar{U}$ (U merged with X); copy the knots on either end which do not change;
4. go into a loop and
   4.1. compute the new control points;
   4.2. merge the elements from U and X into $\bar{U}$;

   the loop can work forward (starting at $\mathbf{Q}_{a-p+1}^w$) or backward (starting at $\mathbf{Q}_{b+r-1}^w$).



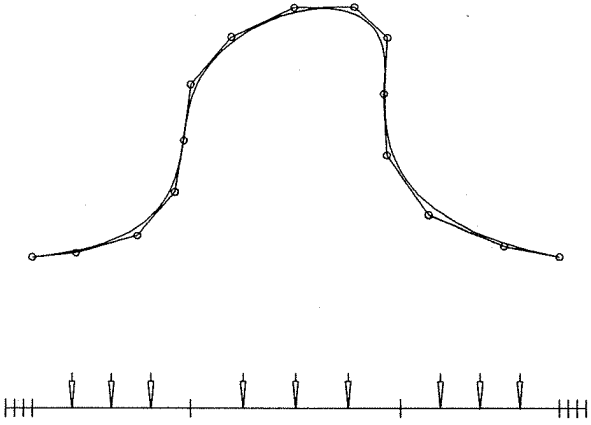
(a)

(b)

Figure 5.16. Curve refinement. (a) Cubic curve defined on $\{0,0,0,0,\frac{3}{10},\frac{7}{10},1,1,1,1\}$; (b) the first midpoint knot refinement; (c) the second midpoint knot refinement.

Algorithm A5.4 is by Boehm and Prautzsch [Boeh85a]. It works backward and overwrites intermediate control points while inserting a knot. Ubar is the new knot vector, $\bar{U}$.

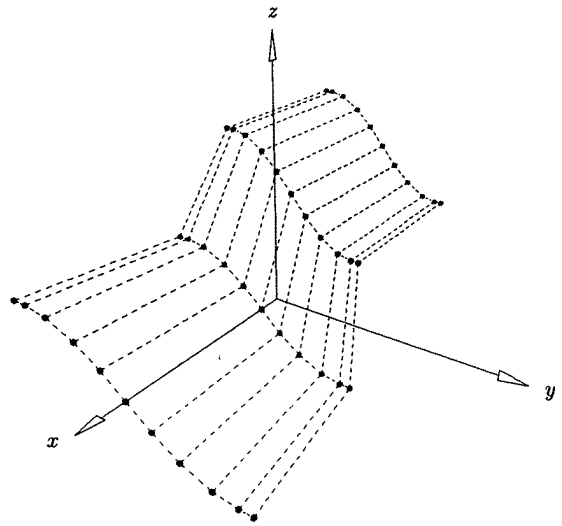
```
ALGORITHM A5.4
  RefineKnotVectCurve(n,p,U,Pw,X,r,Ubar,Qw)
    { /*  Refine curve knot vector  */
      /*  Input:  n,p,U,Pw,X,r  */
```

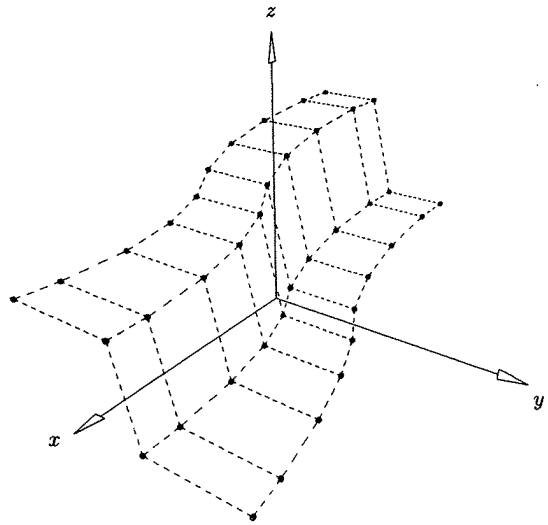
(c)

Figure 5.16. (*Continued.*)

```
      /*  Output: Ubar,Qw  */
m = n+p+1;
a = FindSpan(n,p,X[0],U);    /* Algorithm A2.1 */
b = FindSpan(n,p,X[r],U);
b = b+1;
for (j=0; j<=a-p; j++)    Qw[j] = Pw[j];
for (j=b-1; j<=n; j++)    Qw[j+r+1] = Pw[j];
for (j=0; j<=a; j++)      Ubar[j] = U[j];
for (j=b+p; j<=m; j++)    Ubar[j+r+1] = U[j];
i = b+p-1;     k = b+p+r;
for (j=r; j>=0; j--)
  {
  while (X[j] <= U[i] && i > a)
    {
    Qw[k-p-1] = Pw[i-p-1];
    Ubar[k] = U[i];
    k = k-1;     i = i-1;
    }
  Qw[k-p-1] = Qw[k-p];
  for (l=1; l<=p; l++)
    {
    ind = k-p+l;
    alfa = Ubar[k+l]-X[j];
    if (abs(alfa) == 0.0)
      Qw[ind-1] = Qw[ind];
    else
```

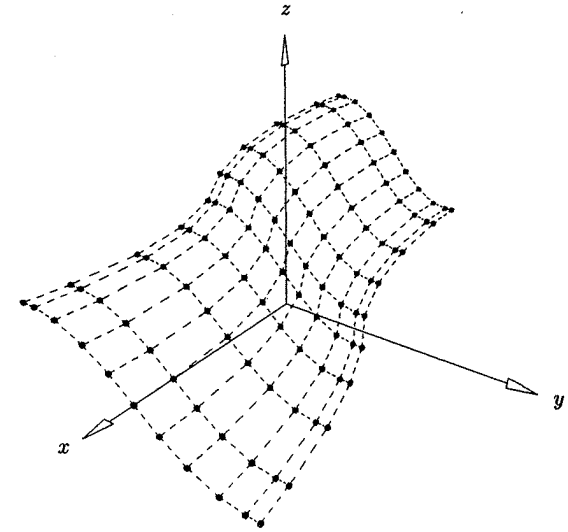


(a)



(b)

Figure 5.17. Refining the surface in Figure 5.9. (a) The third midpoint refinement in the u direction; (b) the second midpoint refinement in the v direction; (c) the third refinement in the u direction and second in the v direction.



(c)

Figure 5.17. (*Continued.*)

```
            {
            alfa = alfa/(Ubar[k+1]-U[i-p+1]);
            Qw[ind-1] = alfa*Qw[ind-1] + (1.0-alfa)*Qw[ind];
            }
          }
        Ubar[k] = X[j];
        k = k-1;
        }
    }
```

Let $\mathbf{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\mathbf{P}_{i,j}^w$ be a NURBS surface on U and V. A U knot vector refinement is accomplished by simply applying Algorithm A5.4 to the $m+1$ columns of control points. A V refinement requires $n+1$ applications of Algorithm A5.3. The algorithm can be organized so that redundant operations are eliminated (e.g., the values of the alfas in the last for-loop are the same for each of the $m+1$ columns of control points). A sketch of the algorithm is:

```
ALGORITHM A5.5
  RefineKnotVectSurface(n,p,U,m,q,V,Pw,X,r,dir,Ubar,Vbar,Qw)
    { /*  Refine surface knot vector  */
      /*  Input:  n,p,U,m,q,V,Pw,X,r,dir  */
      /*  Output: Ubar,Vbar,Qw  */
```

```
if (dir == U_DIRECTION)
  {
  find indexes a and b;
  initialize Ubar;
  copy V into Vbar;
          /* Save unaltered ctrl pts */
  for (row=0; row<=m; row++)
    {
    for (k=0; k<=a-p; k++)  Qw[k][row] = Pw[k][row];
    for (k=b-1; k<=n; k++)  Qw[k+r+1][row] = Pw[k][row];
    }
  for (j=r; j>=0; j--)
    {
    while (X[j]<=U[i] && i>a)
      {
      compute Ubar;
      for ( row ... )  Qw[k-p-1][row] = Pw[i-p-1][row];
      k = k-1;   i = i-1;
      }
    for ( row ... )   Qw[k-p-1][row] = Qw[k-p][row];
    for (l=1; l<=p; l++)
      {
      ind = k-p+l;
      compute alfa;
      if (abs(alfa) == 0.0)
        for ( row ... )   Qw[ind-1][row] = Qw[ind][row];
      else
        {
        compute alfa;
        for ( row ... )
          Qw[ind-1][row] =
            alfa*Qw[ind-1][row]+(1.0-alfa)*Qw[ind][row];
        }
      }
    Ubar[k] = X[j];   k = k-1;
    }
  }
if (dir == V_DIRECTION)
  {
  /* Similar code as above with u and v directional
    parameters switched */
  }
}
```

An important application of knot refinement is the problem of decomposing a NURBS curve into its constituent (four-dimensional) polynomial segments. This

is required when converting a NURBS curve to another spline form, e.g., to the IGES Parametric Spline Curve, Entity type 112 [IGE93]. In such a conversion, the first step is to decompose the curve into its piecewise Bézier form. The Bézier control points of the segments are obtained by inserting each interior knot until it has multiplicity p. This is done in two steps:

1. pass through U and build the refinement vector X;
2. call Algorithm A5.4.

Figures 5.18a and 5.18b show a cubic curve and its corresponding basis functions; Figures 5.19a and 5.19b show the same curve and its basis functions after decomposition. The control points in Figure 5.19a are the Bézier control points of the curve's segments. Figures 5.20a, 5.20b, 5.21a, and 5.21b show the decomposition of a surface.

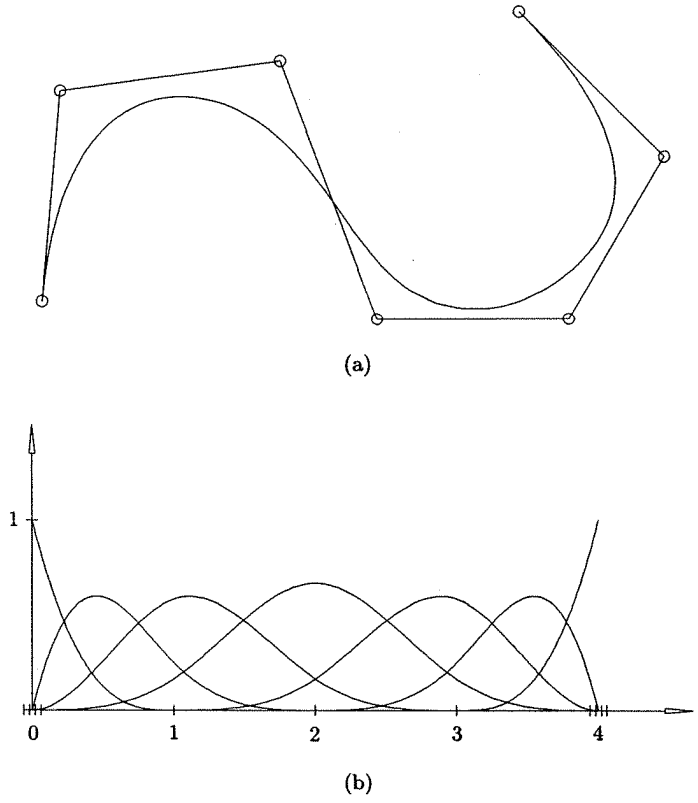
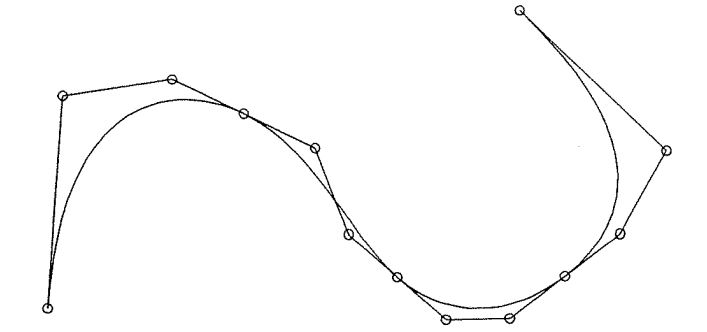
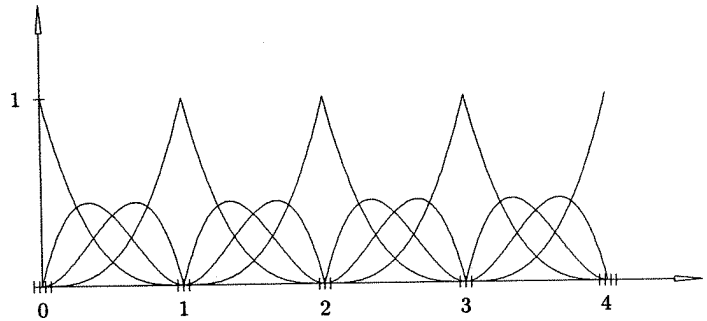


Figure 5.18. A cubic curve. (a) The curve and its control polygon; (b) basis functions defined over $\{0,0,0,0,1,2,3,4,4,4,4\}$.



(a)

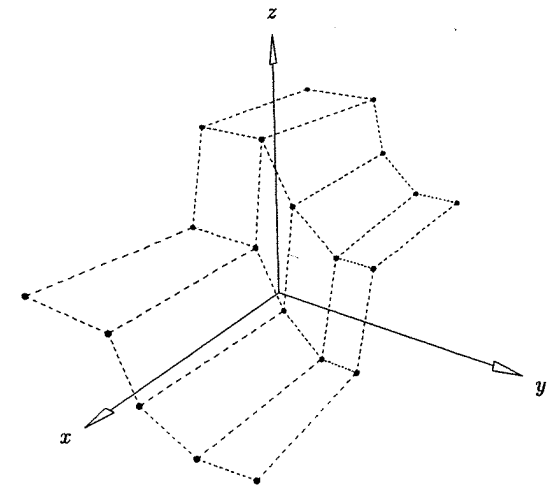


(b)

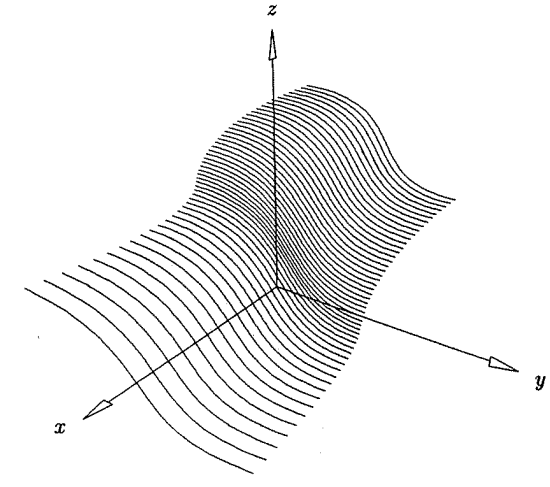
Figure 5.19. Decomposition of the curve in Figure 5.18 into a piecewise Bézier curve via knot refinement. (a) The decomposed curve and its control polygon; (b) basis functions after inserting knots 1, 2, and 3 two times each.

We now give an algorithm which extracts all the Bézier segments of a curve, working from left to right, in an efficient manner. For convenience we drop the w superscipts from control points. Figures 5.22a through 5.22d best explain the algorithm. Let $\mathbf{C}(u) = \sum_{i=0}^{5} N_{i,3}(u)\mathbf{P}_i$ be a cubic curve with two distinct interior knots, and let $\mathbf{Q}_k^j$ be the kth control point of the jth Bézier segment, $k = 0, \ldots, p$ and $j = 0, 1, 2$. Notice in Figures 5.22b and 5.22c that while the rightmost Bézier control points of the zeroth segment, $\mathbf{Q}_2^0$ and $\mathbf{Q}_3^0$, are being computed (via knot insertion), the leftmost Bézier points of the first segment, $\mathbf{Q}_0^1$ and $\mathbf{Q}_1^1$, are also being computed and stored.

Another major reduction in computation, as compared with using general knot refinement, is achieved by examining the computation of the knot insertion alphas (Eq. [5.15]). Assume $[u_a, u_b]$ is the current segment being processed, where a and b are the indices of the rightmost occurrences of knots u_a and u_b.



(a)

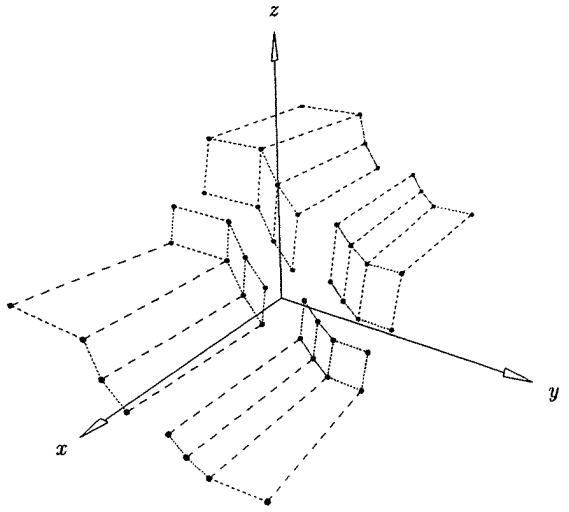


(b)

Figure 5.20. A (cubic × quadratic) surface to be decomposed. (a) The control net; (b) the surface defined over $U = \{0, 0, 0, 0, 3/5, 1, 1, 1, 1\}$ and $V = \{0, 0, 0, 2/5, 1, 1, 1\}$.

Then, when we start to insert u_b the knot vector has the form (locally)

$$\ldots, \underbrace{u_{a-p+1} = \cdots = u_a}_{p}, \underbrace{u_{b-s+1} = \cdots = u_b}_{s} \tag{5.22}$$



(a)

Figure 5.21. Piecewise Bézier patch decomposition via knot refinement. (a) The control nets; (b) Bézier surface patches.

where s is the original multiplicity of u_b. By Eq. (5.15) the insertion of u_b $p-s$ more times generates the triangular table

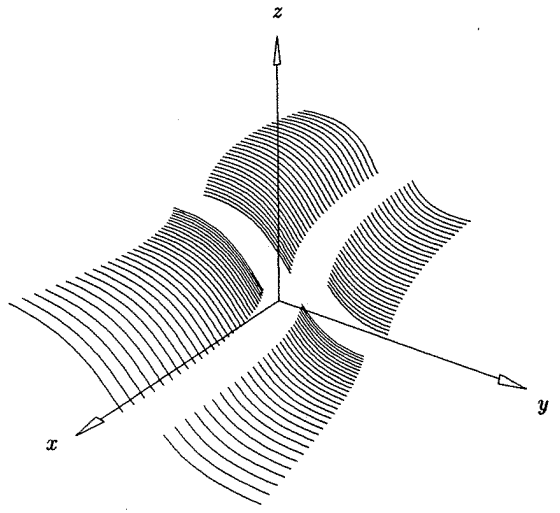
$$\begin{array}{lll} \alpha_{b-p+1,1} & & \\ & \alpha_{b-p+2,2} & \\ \alpha_{b-p+2,1} & & \\ \vdots & \cdots & \alpha_{b-s,p-s} \\ \alpha_{b-s-1,1} & & \\ & \alpha_{b-s,2} & \\ \alpha_{b-s,1} & & \end{array}$$

Examining Eqs. (5.15) and (5.22) reveals two facts:

- $\alpha_{b-p+i,1} = \alpha_{b-p+i+1,2} = \cdots = \alpha_{b-s,p-s-i+1}$ for $i = 1, \ldots, p-s$; the αs are equal along southeasterly pointing diagonals;
- the numerator in Eq. (5.15) remains the same, namely $u_b - u_a$, for the entire set of $\alpha_{i,j}$s.

To illustrate this, consider the example shown in Figure 5.23. After the first segment is extracted, the knot vector takes the form

$$U = \{0,0,0,0,0,\tfrac{1}{5},\tfrac{1}{5},\tfrac{1}{5},\tfrac{1}{5},\tfrac{2}{5},\tfrac{3}{5},\tfrac{4}{5},1,1,1,1,1\}$$



(b)

Figure 5.21. (*Continued.*)

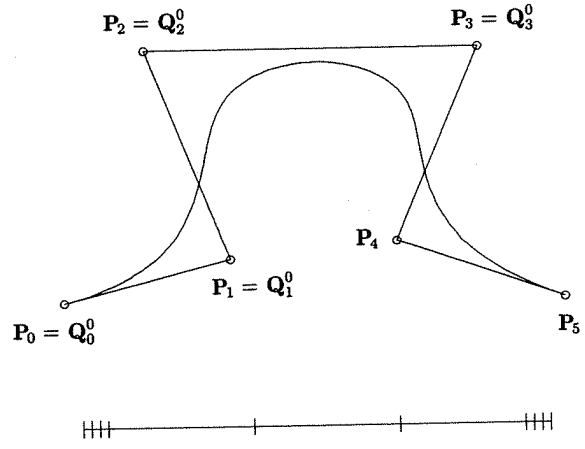
When the second segment is computed, $a = 8$ and $b = 9$, and the following αs are generated

$$\begin{array}{lll} r=1 & r=2 & r=3 \\ \alpha_{6,1} = \dfrac{u_9-u_6}{u_{10}-u_6} = \dfrac{0.2}{0.4} & & \\ & \alpha_{7,2} = \dfrac{u_9-u_7}{u_{10}-u_7} = \dfrac{0.2}{0.4} & \\ \alpha_{7,1} = \dfrac{u_9-u_7}{u_{11}-u_7} = \dfrac{0.2}{0.6} & & \alpha_{8,3} = \dfrac{u_9-u_8}{u_{10}-u_8} = \dfrac{0.2}{0.4} \\ & \alpha_{8,2} = \dfrac{u_9-u_8}{u_{11}-u_8} = \dfrac{0.2}{0.6} & \\ \alpha_{8,1} = \dfrac{u_9-u_8}{u_{12}-u_8} = \dfrac{0.2}{0.8} & & \end{array}$$

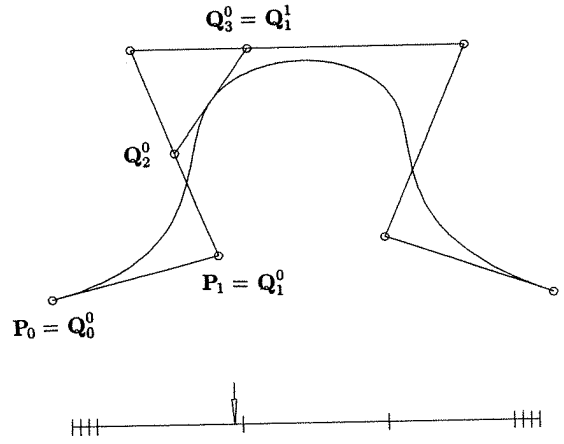
Notice that the αs along southeasterly diagonals are $^{0.2}/_{0.4}$, $^{0.2}/_{0.6}$, and $^{0.2}/_{0.8}$, respectively, and that the numerator remains 0.2.

Algorithm `A5.6` decomposes a NURBS curve and returns `nb` Bézier segments. `Qw[j][k]` is the kth control point of the jth segment. The local array `alphas[]` contains the alphas, with their indices shifted to start at 0.

```
ALGORITHM A5.6
  DecomposeCurve(n,p,U,Pw,nb,Qw)
    { /* Decompose curve into Bézier segments */
```



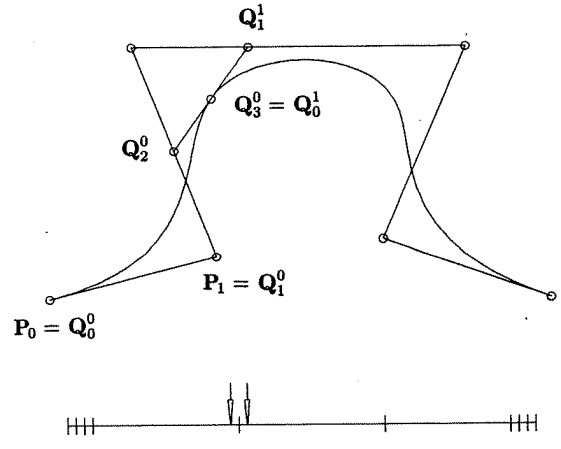
(a)



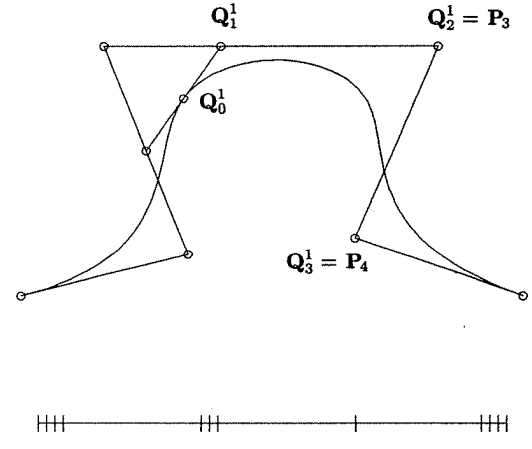
(b)

Figure 5.22. On-the-fly curve decomposition. (a) The original cubic curve; (b) first knot inserted; (c) second knot inserted; (d) preparation for the second segment.

```
   /*  Input:  n,p,U,Pw  */
   /*  Output: nb,Qw  */
m = n+p+1;
a = p;
b = p+1;
nb = 0;
for (i=0; i<=p; i++)    Qw[nb][i] = Pw[i];
```

(c)



(d)

Figure 5.22. (*Continued.*)

```
while (b < m)
  {
  i = b;
  while (b < m && U[b+1] == U[b])   b++;
  mult = b-i+1;
  if (mult < p)
    {
    numer = U[b]-U[a];    /* Numerator of alpha */
```

```
        /* Compute and store alphas */
      for (j=p; j>mult; j--)
        alphas[j-mult-1] = numer/(U[a+j]-U[a]);
      r = p-mult;   /* Insert knot r times */
      for (j=1; j<=r j++)
        {
        save = r-j;
        s = mult+j;   /* This many new points */
        for (k=p; k>=s; k--)
          {
          alpha = alphas[k-s];
          Qw[nb][k] = alpha*Qw[nb][k] + (1.0-alpha)*Qw[nb][k-1];
          }
        if (b < m)          /* Control point of */
          Qw[nb+1][save] = Qw[nb][p];  /* next segment */
        }
      }
    nb = nb+1;   /* Bézier segment completed */
    if (b < m)
      {   /* Initialize for next segment */
      for (i=p-mult; i<=p; i++)   Qw[nb][i] = Pw[b-p+i];
      a = b;
      b = b+1;
      }
    }
  }
```

Algorithm A5.6 is an ideal interface between a NURBS system and a system (hardware or software) which displays Bézier curves. With modifications, Algorithm A5.6 computes the Bézier segments on-the-fly, passes each one down for display, and overwrites each with the next. Nothing is returned. These modifications are:

- remove nb and Qw from the argument list;
- use local arrays Qw[] (length $p+1$) and NextQw[] (length $p-1$); NextQw is used to store the leftmost control points of the next segment;
- after a segment has been computed (see the comment in the code, /* Bézier segment completed */) pass it down for display;
- after displaying a segment, overwrite it with points from NextQw[] and Pw[].

An example is shown in Figures 5.24a–5.24c; for more detail see [Pieg91b].

Algorithm A5.7 shows the organization of a surface decomposition. The routine computes a Bézier strip, i.e., a NURBS surface that is Bézier in one direction and B-spline in the other. The routine must be called twice, once in the u direction to get the Bézier strips, and then the strips must be fed into the routine in the v direction to get the Bézier patches.

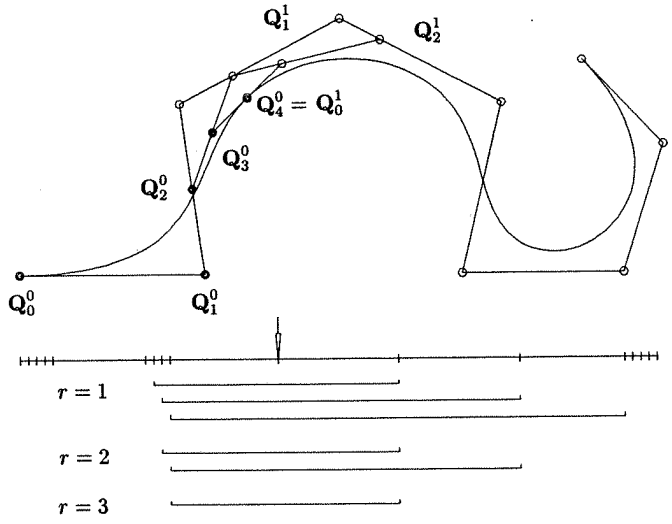
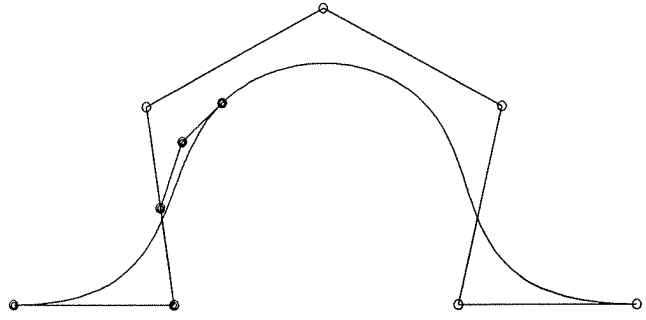


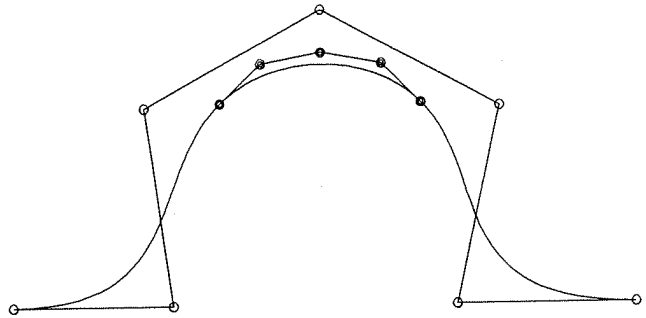
Figure 5.23. Ratios for the alphas used to compute the second Bézier segment of a quartic curve.

```
ALGORITHM A5.7
  DecomposeSurface(n,p,U,m,q,V,Pw,dir,nb,Qw)
    { /*  Decompose surface into Bézier patches  */
      /*  Input:  n,p,U,m,q,V,Pw,dir  */
      /*  Output: nb,Qw  */
    if (dir == U_DIRECTION)
```

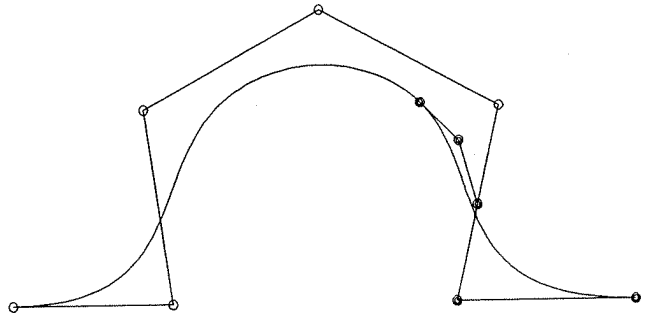


(a)

Figure 5.24. On-the-fly decomposition of a quartic curve. (a) The first Bézier segment; (b) the second Bézier segment; (c) the third Bézier segment.



(b)



(c)

Figure 5.24. (*Continued.*)

```
{
a=p;    b=p+1;
nb=0;
for (i=0; i<=p; i++)
  for (row=0; row<=m; row++)
    Qw[nb][i][row] = Pw[i][row];
while (b<m)
  {
  get mult;
  if ( mult<p )
    {
    get the numerator and the alfas;
    for (j=1; j<=p-mult; j++)
      {
      save= ...;
      s = ...;
```



```
      for(k=p; k>=s; k--)
        {
        get alfa;
        for ( row ... )
          Qw[nb][k][row] = alfa*Qw[nb][k][row]
                           +(1.0-alfa)*Qw[nb][k-1][row];
        }
      if ( b<m )
        for ( row ... )
          Qw[nb+1][save][row] = Qw[nb][p][row];
      }
    }
  nb=nb+1;
  if ( b<m )
    {
    for (i=p-mult; i<=p; i++)
      for ( row ... )
        Qw[nb][i][row] = Pw[b-p+i][row];
    a = b;    b = b+1;
    }
  }
}
if (dir == V_DIRECTION)
  {
  /* Similar code as above with u- and v-directional
    parameters switched */
  }
}
```

An example is shown in Figure 5.25.

5.4 Knot Removal

Knot removal is the reverse process of knot insertion. Let

$$\mathbf{C}^w(u) = \sum_{i=0}^{n} N_{i,p}(u)\,\mathbf{P}_i^w \tag{5.23}$$

be defined on U, and let u_r be an interior knot of multiplicity s in U; end knots are not removed. Let U_t denote the knot vector obtained by removing u_r t times from U $(1 \le t \le s)$. We say that u_r is t times *removable* if $\mathbf{C}^w(u)$ has a precise representation of the form

$$\mathbf{C}^w(u) = \sum_{i=0}^{n-t} \bar{N}_{i,p}(u)\,\mathbf{Q}_i^w \tag{5.24}$$

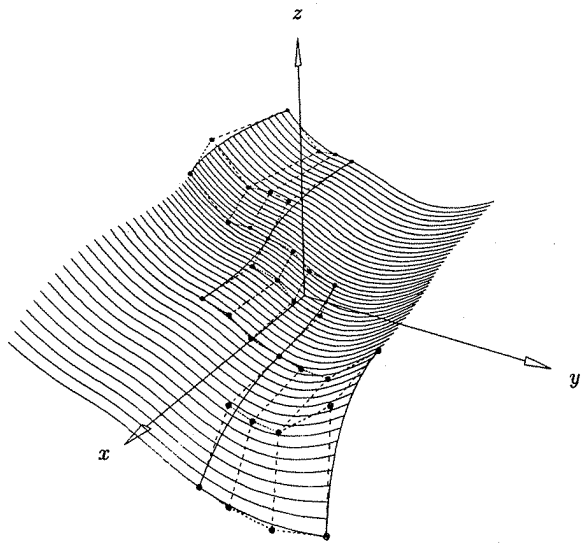


Figure 5.25. Surface decomposition on the fly; three Bézier patches are shown.

where $\bar{N}_{i,p}(u)$ are the basis functions on U_t, that is, Eqs. (5.23) and (5.24) geometrically and parametrically represent the same curve.

From earlier chapters we know that the basis functions $N_{i,p}(u)$ which are nonzero at u_r are only C^{p-s} continuous there. Furthermore, although one does not generally expect more than C^{p-s} continuity of the curve, the control points $\mathbf{P}_i^w$ can lie in positions such that the $(p-s+1)$th (or even higher) derivative is continuous. Hence, the knot u_r is t times removable if and only if the curve $\mathbf{C}^w(u)$ is C^{p-s+t} continuous at $u = u_r$. It is important to note that the continuity must be with respect to $\mathbf{C}^w(u)$, not its projection $\mathbf{C}(u)$ which can be continuous even though $\mathbf{C}^w(u)$ is not.

A knot removal algorithm must do two things:

- determine if a knot is removable and how many times;
- compute the new control points, $\mathbf{Q}_i^w$.

Details can be found in [Till92].

Knot removal is an important utility in several applications:

- The standard method to convert a spline curve or surface represented in power basis form to B-spline form is:
  - convert the segments (patches) to Bézier form;
  - obtain a B-spline representation by piecing the Bézier segments together and using knot vectors in which all interior knots have multiplicity equal to the degree;
  - remove as many knots (and hence control points) as the continuity of the spline curve (surface) allows;
- When interactively shaping B-spline curves and surfaces, knots are sometimes added to increase the number of control points which can be modified. When control points are moved, the level of continuity at the knots can change (increase or decrease); hence, after modification is completed knot removal can be invoked in order to obtain the most compact representation of the curve or surface;
- It is sometimes useful to link B-spline curves together to form composite curves. The first step is to make the curves compatible, i.e., of common degree, and the end parameter value of the ith curve is equal to the start parameter of the $(i+1)$th curve. Once this is done, the composition is accomplished by using interior knots of multiplicity equal to the common degree of the curves. Knot removal can then be invoked in order to remove unnecessary knots.

We describe the knot removal process with an example. As usual, the algorithm operates on the four coordinates of the control points, but we drop the w superscript for the sake of notational convenience. Consider the cubic curve of Figure 5.26. Assume the original curve is defined by $\{\mathbf{P}_0^0, \ldots, \mathbf{P}_6^0\}$ and $U = \{u_0, \ldots, u_{10}\}$, where the superscript on the control points denotes the step number in the knot removal process, and the knots are $u_0 = \cdots = u_3 = 0$, $u_4 = u_5 = u_6 = 1$, and $u_7 = \cdots = u_{10} = 2$. Consider removing $u = 1 (= u_6)$.

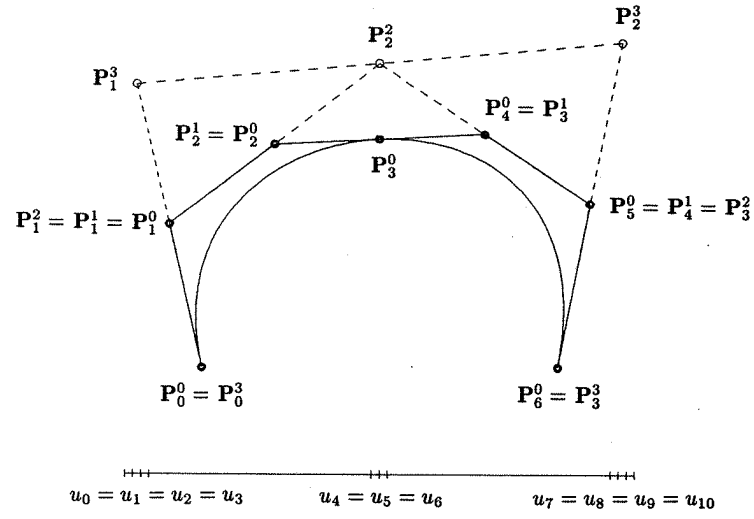


Figure 5.26. Knot removal from a cubic curve with a triple knot.

The first derivative is continuous if and only if

$$\frac{p}{u_6 - u_3}(\mathbf{P}_3^0 - \mathbf{P}_2^0) = \frac{p}{u_7 - u_4}(\mathbf{P}_4^0 - \mathbf{P}_3^0)$$

Setting $u = u_4 = u_6$ and rearranging terms yields

$$\mathbf{P}_3^0 = \frac{u - u_3}{u_7 - u_3}\mathbf{P}_4^0 + \frac{u_7 - u}{u_7 - u_3}\mathbf{P}_2^0$$

Using $\mathbf{P}_2^0 = \mathbf{P}_2^1$ and $\mathbf{P}_4^0 = \mathbf{P}_3^1$ yields

$$\mathbf{P}_3^0 = \alpha_3\mathbf{P}_3^1 + (1 - \alpha_3)\mathbf{P}_2^1 \qquad \alpha_3 = \frac{u - u_3}{u_7 - u_3} \tag{5.25}$$

Equation (5.25) is just the equation for inserting $u = 1$ into the knot vector having 1 as a double knot (u_7 is u_6 in that knot vector). Furthermore, the second derivative at $u = 1$ is continuous if and only if the knot $u = 1$ can be removed a second time. It can be removed when

$$\begin{aligned}\mathbf{P}_2^1 &= \alpha_2\mathbf{P}_2^2 + (1 - \alpha_2)\mathbf{P}_1^2\\ \mathbf{P}_3^1 &= \alpha_3\mathbf{P}_3^2 + (1 - \alpha_3)\mathbf{P}_2^2\end{aligned} \tag{5.26}$$

with $$\alpha_i = \frac{u - u_i}{u_{i+p+2} - u_i} \qquad i = 2, 3$$

Finally, the third derivative is continuous if and only if $u = 1$ can be removed a third time, which implies that

$$\begin{aligned}\mathbf{P}_1^2 &= \alpha_1\mathbf{P}_1^3 + (1 - \alpha_1)\mathbf{P}_0^3\\ \mathbf{P}_2^2 &= \alpha_2\mathbf{P}_2^3 + (1 - \alpha_2)\mathbf{P}_1^3\\ \mathbf{P}_3^2 &= \alpha_3\mathbf{P}_3^3 + (1 - \alpha_3)\mathbf{P}_2^3\end{aligned} \tag{5.27}$$

with $$\alpha_i = \frac{u - u_i}{u_{i+p+3} - u_i} \qquad i = 1, 2, 3$$

Let $n = p - s + 1$, where s is the multiplicity of the knot to be removed. Then note that

- there are no unknowns in Eq. (5.25), $\mathbf{P}_2^2$ is the only unknown in Eq. (5.26), and $\mathbf{P}_1^3$ and $\mathbf{P}_2^3$ are unknown in Eq. (5.27). Knot removal produces n equations in $n - 1$ unknowns;
- knot removal destroys n control points and replaces them with $n - 1$ new control points.

Let us solve Eqs. (5.25) through (5.27). For Eq. (5.25) we compute the right side of the single equation and check to see if it is equal to $\mathbf{P}_3^0$ (to within a meaningful tolerance). If not equal the knot cannot be removed; if it is equal, then the knot and $\mathbf{P}_3^0$ are removed.

From the two equations in Eq. (5.26) we obtain

$$\mathbf{P}_2^2 = \frac{\mathbf{P}_2^1 - (1 - \alpha_2)\mathbf{P}_1^2}{\alpha_2} \qquad \mathbf{P}_2^2 = \frac{\mathbf{P}_3^1 - \alpha_3\mathbf{P}_3^2}{1 - \alpha_3}$$

If these two values for $\mathbf{P}_2^2$ are within tolerance, then the knot can be removed and $\mathbf{P}_2^1$ and $\mathbf{P}_3^1$ are replaced by $\mathbf{P}_2^2$.

Solving the first and third equations of Eq. (5.27), we obtain

$$\mathbf{P}_1^3 = \frac{\mathbf{P}_1^2 - (1 - \alpha_1)\mathbf{P}_0^3}{\alpha_1} \qquad \mathbf{P}_2^3 = \frac{\mathbf{P}_3^2 - \alpha_3\mathbf{P}_3^3}{1 - \alpha_3}$$

Then substitute $\mathbf{P}_1^3$ and $\mathbf{P}_2^3$ into the right side of the second of Eqs. (5.27), and check to see if it is within tolerance of $\mathbf{P}_2^2$. If it is, remove the knot and replace $\mathbf{P}_1^2$, $\mathbf{P}_2^2$, and $\mathbf{P}_3^2$ with $\mathbf{P}_1^3$ and $\mathbf{P}_2^3$.

In general, to solve the system of equations start with the first and last equations and work inward, computing the new control points. If the number of equations is even (as in Eq. [5.26]), then the final new control point is computed twice. The knot can be removed if the two values of the control point are within tolerance. If the number of equations is odd (as in Eqs. [5.25] and [5.27]), then all new control points are computed once, and the last two computed are substituted into the middle equation. If the result is within tolerance of the old control point on the left side of the middle equation, then the knot is removable.

We now give general formulas. Let $u = u_r \neq u_{r+1}$ be a knot of multiplicity s, where $1 \leq s \leq p$. The equations for computing the new control points for one removal of u are

$$\begin{aligned}\mathbf{P}_i^1 &= \frac{\mathbf{P}_i^0 - (1 - \alpha_i)\mathbf{P}_{i-1}^1}{\alpha_i} \qquad r - p \leq i \leq \frac{1}{2}(2r - p - s - 1)\\ \mathbf{P}_j^1 &= \frac{\mathbf{P}_j^0 - \alpha_j\mathbf{P}_{j+1}^1}{(1 - \alpha_j)} \qquad \frac{1}{2}(2r - p - s + 2) \leq j \leq r - s\end{aligned} \tag{5.28}$$

with $$\alpha_k = \frac{u - u_k}{u_{k+p+1} - u_k} \qquad k = i, j$$

The simplest way to program Eq. (5.28) is

```
i = r - p;  j = r - s;
while (j - i > 0)
    α_i = ···
    α_j = ···
    P_i^1 = ···
    P_j^1 = ···
    i = i + 1;  j = j - 1;
```

Now suppose we want to remove $u = u_r$ multiple times. Each time the knot is removed, s and r are decremented (in Eq. [5.28]) and the superscripts on the control points are incremented. Thus, the equations for removing $u = u_r$ the tth time are

$$\mathbf{P}_i^t = \frac{\mathbf{P}_i^{t-1} - (1-\alpha_i)\,\mathbf{P}_{i-1}^t}{\alpha_i} \qquad r-p-t+1 \le i \le \frac{1}{2}(2r-p-s-t)$$

$$\mathbf{P}_j^t = \frac{\mathbf{P}_j^{t-1} - \alpha_j \mathbf{P}_{j+1}^t}{(1-\alpha_j)} \qquad \frac{1}{2}(2r-p-s+t+1) \le j \le r-s+t-1 \tag{5.29}$$

with $$\alpha_i = \frac{u-u_i}{u_{i+p+t}-u_i} \qquad \alpha_j = \frac{u-u_{j-t+1}}{u_{j+p+1}-u_{j-t+1}}$$

Assume u is to be removed k times. Equation (5.29) can be programmed as

```
first = r-p+1;  last = r-s-1;
for t = 1,...,k
  first = first - 1;  last = last + 1;
  i = first;  j = last;
  while (j - i > t - 1)
    α_i = ···
    α_j = ···
    P_i^t = ···
    P_j^t = ···
    i = i + 1;  j = j - 1;
```

The factor which complicates the implementation of knot removal is that it is generally unknown in advance whether a knot is removable, and if it is, how many times. The (potentially) new control points must be computed in temporary storage, and it is not known in advance how many knots and control points will be in the output. Algorithm `A5.8` tries to remove the knot $u = u_r$ *num* times, where $1 \le num \le s$. It returns t, the actual number of times the knot is removed, and if $t > 0$ it returns the new knot vector and control points. It computes the new control points in place, overwriting the old ones. Only one local array (`temp`) of size $2p+1$ is required to compute the new control points at each step. If removal fails at step t, the control points from step $t-1$ are still intact. At the end, knots and control points are shifted down to fill the gap left by removal. A minimum number of shifts is done. To check for coincident points, a value, `TOL`, and a function, `Distance4D()`, which computes the distance between the points, are used. If the curve is rational we assume the homogeneous representation; in this case, `Distance4D()` computes the four-dimensional distance. The weights are treated as ordinary coordinates. `TOL` has the meaning [Till92]

- if the curve is nonrational (all $w_i = 1$), then one knot removal results in a curve whose deviation from the original curve is less than `TOL`, on the entire parameter domain;
- if the curve is rational, then the deviation is everywhere less than

$$\frac{\mathtt{TOL}(1+|\mathbf{P}|_{\max})}{w_{\min}}$$

where $w_{\min}$ is the minimum weight on the original curve, and $|\mathbf{P}|_{\max}$ is the maximum distance of any three-dimensional point on the original curve from the origin. The convex hull property of B-splines is used to compute bounds for $w_{\min}$ and $|\mathbf{P}|_{\max}$. If the desired bound on deviation is d, then `TOL` should be set to

$$\mathtt{TOL} = \frac{d\,w_{\min}}{1+|\mathbf{P}|_{\max}} \tag{5.30}$$

```
ALGORITHM A5.8
RemoveCurveKnot(n,p,U,Pw,u,r,s,num,t)
  { /*  Remove knot u (index r) num times.  */
    /*  Input:  n,p,U,Pw,u,r,s,num  */
    /*  Output: t, new knots & ctrl pts in U & Pw  */
  m = n+p+1;
  ord = p+1;
  fout = (2*r-s-p)/2;  /* First control point out */
  last = r-s;
  first = r-p;
  for (t=0; t<num; t++)
    { /* This loop is Eq.(5.28) */
    off = first-1;  /* Diff in index between temp and P */
    temp[0] = Pw[off];   temp[last+1-off] = Pw[last+1];
    i = first;      j=last;
    ii = 1;      jj = last-off;
    remflag = 0;
    while (j-i > t)
      { /* Compute new control points for one removal step */
      alfi = (u-U[i])/(U[i+ord+t]-U[i]);
      alfj = (u-U[j-t])/(U[j+ord]-U[j-t]);
      temp[ii] = (Pw[i]-(1.0-alfi)*temp[ii-1])/alfi;
      temp[jj] = (Pw[j]-alfj*temp[jj+1])/(1.0-alfj);
      i = i+1;     ii = ii+1;
      j = j-1;     jj = jj-1;
      } /* End of while-loop */
    if (j-i < t)  /* Check if knot removable */
      {
      if (Distance4D(temp[ii-1],temp[jj+1]) <= TOL)
        remflag = 1;
```

```
    }
  else
    {
    alfi = (u-U[i])/(U[i+ord+t]-U[i]);
    if (Distance4D(Pw[i],alfi*temp[ii+t+1]
                          +(1.0-alfi)*temp[ii-1]) <= TOL)
      remflag = 1;
    }
  if (remflag == 0)   /* Cannot remove any more knots */
    break;            /* Get out of for-loop */
  else
    {   /* Successful removal. Save new cont.pts. */
    i = first;   j = last;
    while (j-i > t)
      {
      Pw[i] = temp[i-off];   Pw[j] = temp[j-off];
      i = i+1;    j = j-1;
      }
    }
  first = first-1;    last = last+1;
  } /* End of for-loop */
if (t == 0)   return;
for (k=r+1; k<=m; k++) U[k-t] = U[k];  /* Shift knots */
j = fout;   i=j;   /* Pj thru Pi will be overwritten */
for (k=1; k<t; k++)
  if (Mod(k,2) == 1)    /* k modulo 2 */
    i = i+1;    else    j = j-1;
for (k=i+1; k<=n; k++)    /* Shift */
  { Pw[j] = Pw[k];   j = j+1;  }
return;
}
```

Table 5.2 shows how the control point array changes for the cubic curve example given in Figure 5.26. The first row shows the contents before entering the for-loop ($0 \le t < num$). Rows 2–4 show the array (changes only) at the bottom of the for-loop for $t = 0$, $t = 1$, and $t = 2$. An 'X' denotes an unused array element.

We remark that Algorithm A5.8 can create negative or zero weights. Theoretically this is correct, but it may be undesirable for software reasons. It can be avoided by simply inserting a check after the distance computation, before setting remflag to 1.

Let $\mathbf{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\mathbf{P}^w_{i,j}$ be a NURBS surface. A u knot (v knot) is removed from $\mathbf{S}^w(u,v)$ by applying the knot removal algorithm (Algorithm A5.8) to the $m+1$ columns ($n+1$ rows) of control points. But the knot can be removed only if the removal is successful for all $m+1$ columns ($n+1$ rows).

Two additional algorithms for knot removal are given in [Till92]. These are:

- given a curve, remove as many knots as possible;

Table 5.2. The control point array for the cubic curve of Figure 5.15.

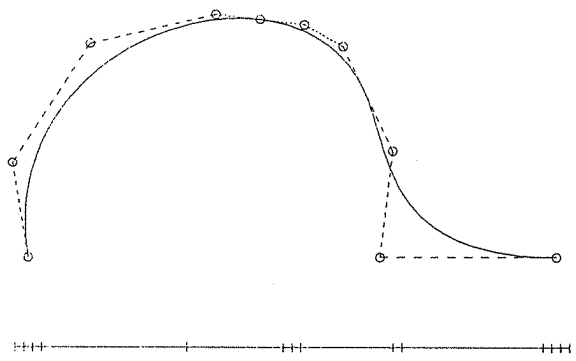
$\mathbf{P}_0^0$	$\mathbf{P}_1^0$	$\mathbf{P}_2^0$	$\mathbf{P}_3^0$	$\mathbf{P}_4^0$	$\mathbf{P}_5^0$	$\mathbf{P}_6^0$	before loop
			X				$t=0$
		$\mathbf{P}_2^2$	X	$\mathbf{P}_2^2$			$t=1$
	$\mathbf{P}_1^3$	X	X	X	$\mathbf{P}_2^3$		$t=2$

- given a surface, remove as many u knots as possible (v knots are analogous).

Instead of giving details of these algorithms, which are quite involved, we illustrate the effects of curve and surface knot removal through a variety of examples. Figure 5.27a shows a curve with single, double, and triple knots. In Figure 5.27b the single knot, $u = 3/10$, and in Figures 5.27c–5.27e one, two, and three occurrences of the triple knot, $u = 1/2$, respectively, are removed.

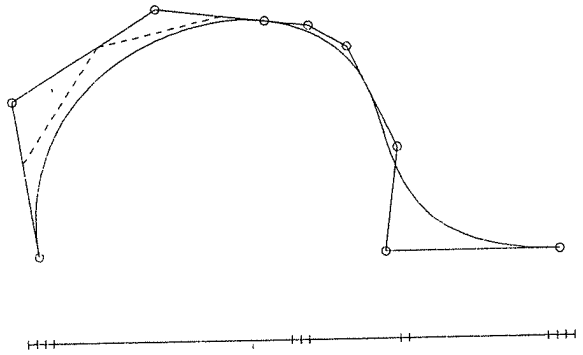
Surface knot removal examples are shown in Figures 5.28a–5.28d. Figure 5.28a shows the original surface. In Figure 5.28b the triple knot, $u = 3/10$, is removed three times. In Figure 5.28c the single knot, $v = 1/4$, is removed, and in Figure 5.28d both knots are removed at the same time. In Figures 5.27b–5.27d as well as in Figures 5.28b–5.28d the original curve/surface is shown dashed, whereas the knot removed curves/surfaces are drawn solid.

All removable knots are removed from the curve illustrated in Figure 5.29a. The curves shown in Figures 5.29b–5.29f were computed using the tolerance values 0.007, 0.025, 0.07, 0.6, and 1.2, respectively. These figures illustrate how many and what knots are removable (removed knots are circled and the original polygon and curve are drawn dashed); how to compute the new control

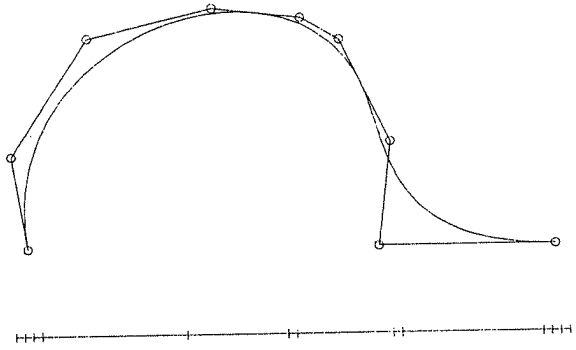


(a)

Figure 5.27. Curve knot removal example. (a) The original cubic curve defined over the knot vector $\{0,0,0,0,3/10,1/2,1/2,1/2,7/10,7/10,1,1,1,1\}$; (b) removal of $3/10$, one time; (c) removal of $1/2$ one time; (d) removal of $1/2$ two times; (e) removal of $1/2$ three times.



(b)



(c)

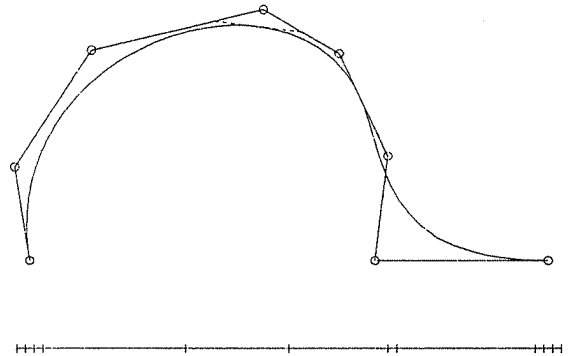
Figure 5.27. (*Continued.*)

points; and what is the deviation between the original and the knot removed curve. Similar examples are shown for the surfaces in Figures 5.30a and 5.30b through Figures 5.34a and 5.34b. The control net for the original surface is seen in Figure 5.30a. The original surface is shown in Figure 5.30b. Knot removed surfaces are illustrated in Figures 5.31a and 5.31b through Figures 5.34a and 5.34b, using the tolerances 0.05, 0.1, 0.3, and 0.5, respectively.

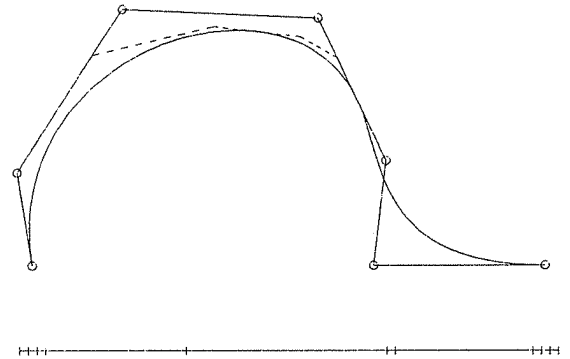
5.5 Degree Elevation

Let $\mathbf{C}_n(u) = \sum_{i=0}^{n} \mathbf{a}_i u^i$ be an nth-degree polynomial curve. Clearly, we can set $\mathbf{a}_{n+1} = 0$ and write $\mathbf{C}_n(u)$ as an $(n+1)$th-degree curve

$$\mathbf{C}_n(u) = \mathbf{C}_{n+1}(u) = \sum_{i=0}^{n+1} \mathbf{a}_i u^i$$



(d)



(e)

Figure 5.27. (*Continued.*)

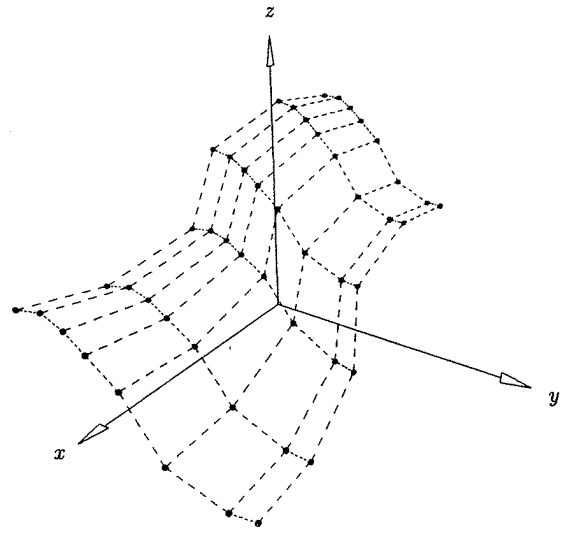
From a vector space point of view, $\mathbf{C}_{n+1}(u)$ is simply $\mathbf{C}_n(u)$ embedded in a higher dimensional space.

Now let $\mathbf{C}_p^w(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i^w$ be a pth-degree NURBS curve on the knot vector U. Since $\mathbf{C}_p^w(u)$ is a piecewise polynomial curve, it should be possible to elevate its degree to $p+1$, that is, there must exist control points $\mathbf{Q}_i^w$ and a knot vector $\hat{U}$ such that

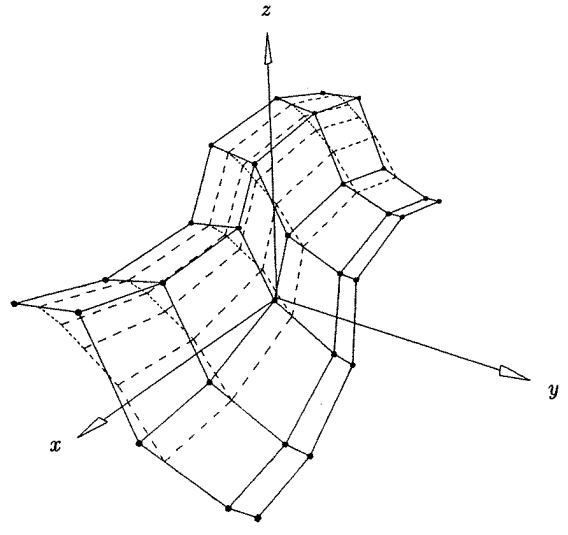
$$\mathbf{C}_p^w(u) = \mathbf{C}_{p+1}^w(u) = \sum_{i=0}^{\hat{n}} N_{i,p+1}(u)\,\mathbf{Q}_i^w \tag{5.31}$$

$\mathbf{C}_{p+1}^w(u)$ and $\mathbf{C}_p^w(u)$ are the same curve, both geometrically and parametrically. $\mathbf{C}_{p+1}^w(u)$ is simply $\mathbf{C}_p^w(u)$ embedded in a higher dimensional space. *Degree elevation* refers to the process (the algorithm) for computing the unknown $\mathbf{Q}_i^w$ and $\hat{U}$.

The applications of degree elevation include:

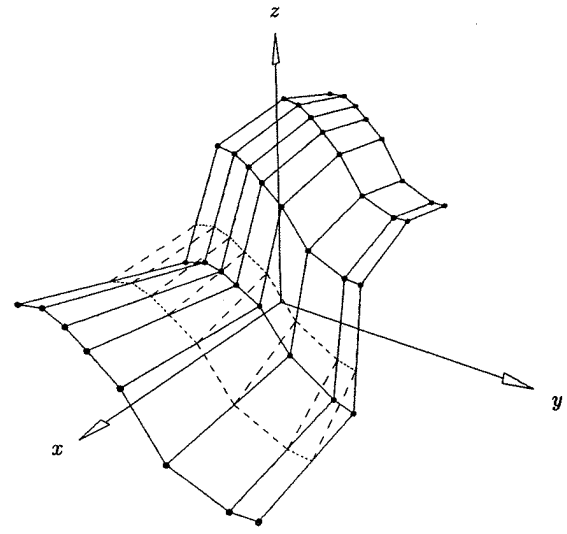


(a)

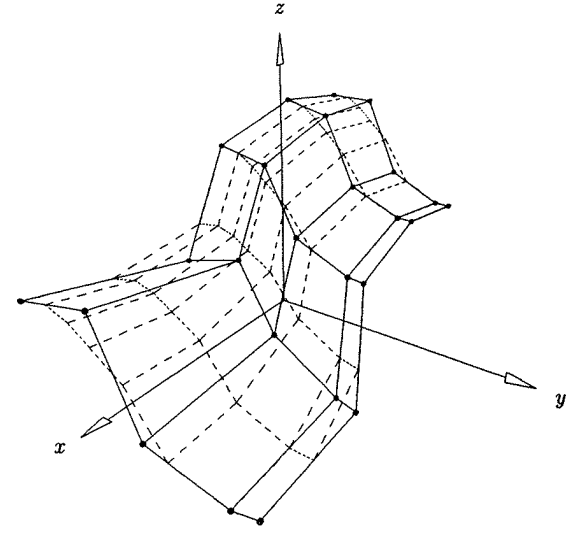


(b)

Figure 5.28. Surface knot removal example. (a) The original (cubic × quadratic) surface defined over $U = \{0,0,0,0,\frac{3}{10},\frac{3}{10},\frac{3}{10},\frac{7}{10},1,1,1,1\}$ and $V = \{0,0,0,\frac{1}{4},\frac{1}{2},\frac{3}{4},1,1,1\}$; (b) removal of $u = \frac{3}{10}$ three times in the u direction.

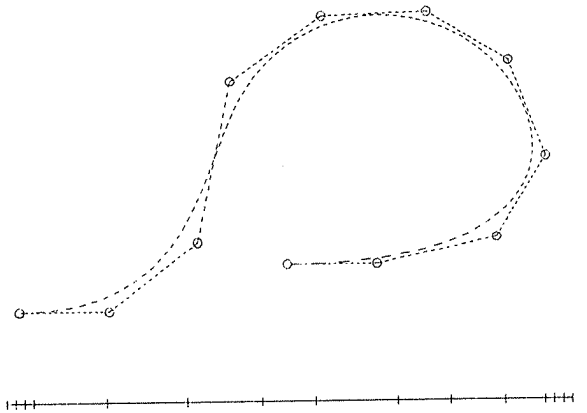


(c)

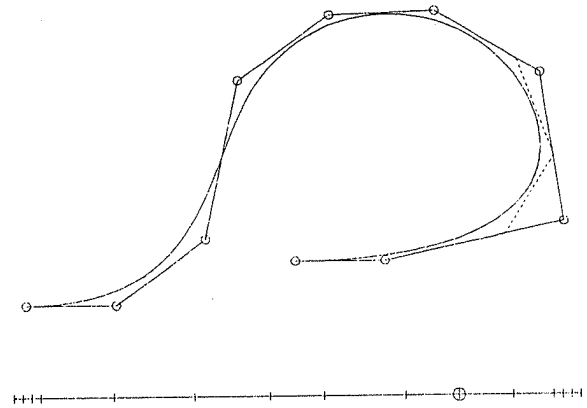


(d)

Figure 5.28. *Continued.* (c) removal of $v = \frac{1}{4}$ one time in the v direction; (d) removal of the knots in both directions.



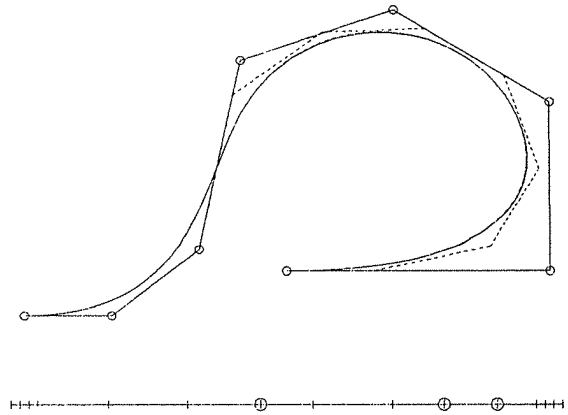
(a)



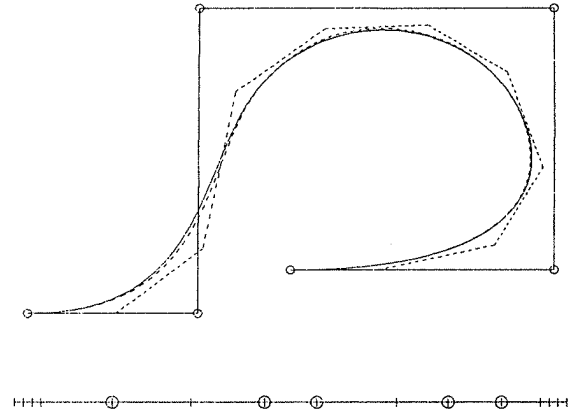
(b)

Figure 5.29. Remove all removable knots from a cubic curve. (a) The original curve defined over $\{0,0,0,0,0.16,0.31,0.45,0.5501,0.702,0.8,0.901,1,1,1,1\}$; (b) knot removed curve using the tolerance 0.007; (c) tolerance is 0.025; (d) tolerance is 0.07; (e) tolerance is 0.6; (f) tolerance is 1.2.

- in subsequent chapters we construct certain types of surfaces from a set of curves $\mathbf{C}_1,\ldots,\mathbf{C}_n$, $(n \geq 2)$. Using tensor product surfaces requires that these curves have a common degree, hence the degrees of some curves may require elevation;
- let $\mathbf{C}_1,\ldots,\mathbf{C}_n$, $(n \geq 2)$ be a sequence of NURBS curves with the property that the endpoint of $\mathbf{C}_i$ is coincident with the starting point of $\mathbf{C}_{i+1}$; then the curves can be combined into a single NURBS curve. One step in this process is to elevate the curves to a common degree.



(c)



(d)

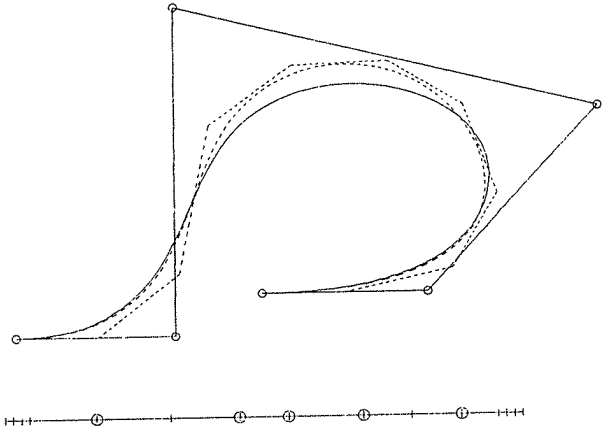
Figure 5.29. (*Continued.*)

As usual, degree elevation algorithms are applied to $\mathbf{C}_p^w(u)$ in four-dimensional space, not to $\mathbf{C}_p(u)$, and again we drop the w for the remainder of this section.

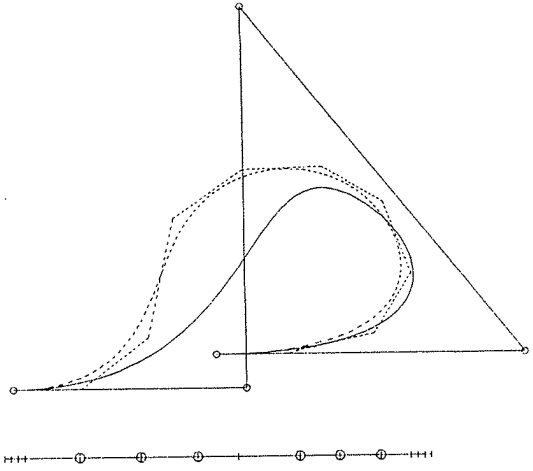
There are three unknowns in Eq. (5.31), $\hat{n}$, $\hat{U}$, and the $\{\mathbf{Q}_i\}$. To determine $\hat{n}$ and $\hat{U}$ assume that U has the form

$$U = \{u_0,\ldots,u_m\} = \{\underbrace{a,\ldots,a}_{p+1},\underbrace{u_1,\ldots,u_1}_{m_1},\ldots,\underbrace{u_s,\ldots,u_s}_{m_s},\underbrace{b,\ldots,b}_{p+1}\}$$

where $m_1,\ldots,m_s$ denote the multiplicities of the interior knots. Now $\mathbf{C}_p(u)$ is a polynomial curve on each nondegenerate knot span, hence its degree can be elevated to $p+1$ on each such knot span. At a knot of multiplicity m_i, $\mathbf{C}_p(u)$ is C^{p-m_i} continuous. Since the degree elevated curve, $\mathbf{C}_{p+1}(u)$, must have the



(e)



(f)

Figure 5.29. (*Continued.*)

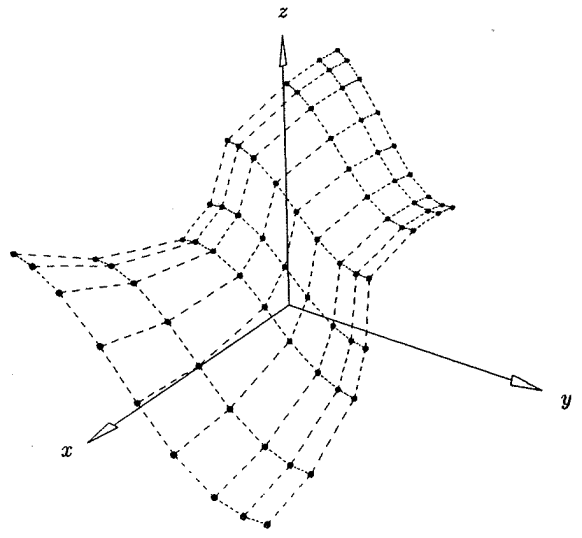
same continuity, it follows that the same knot must have multiplicity $m_i + 1$ for $\mathbf{C}_{p+1}(u)$. This yields

$$\hat{n} = n + s + 1 \tag{5.32}$$

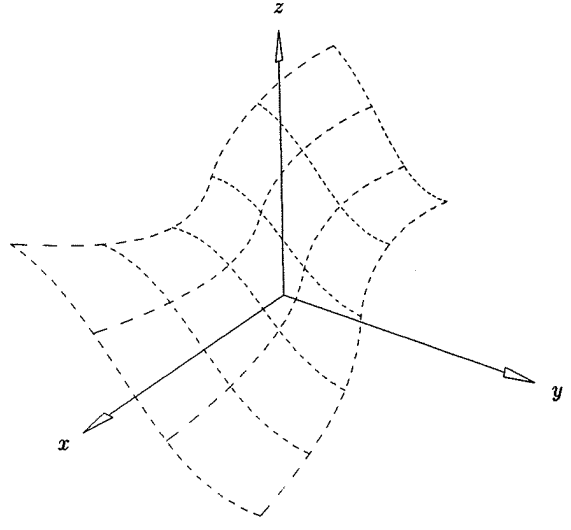
and

$$\hat{U} = \{u_0, \ldots, u_{\hat{m}}\} = \{\underbrace{a, \ldots, a}_{p+2}, \underbrace{u_1, \ldots, u_1}_{m_1+1}, \ldots, \underbrace{u_s, \ldots, u_s}_{m_{s+1}}, \underbrace{b, \ldots, b}_{p+2}\} \tag{5.33}$$

where $\hat{m} = m + s + 2$.

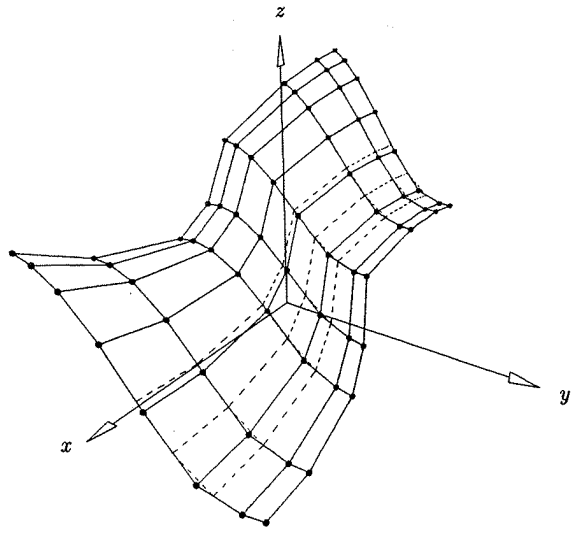


(a)

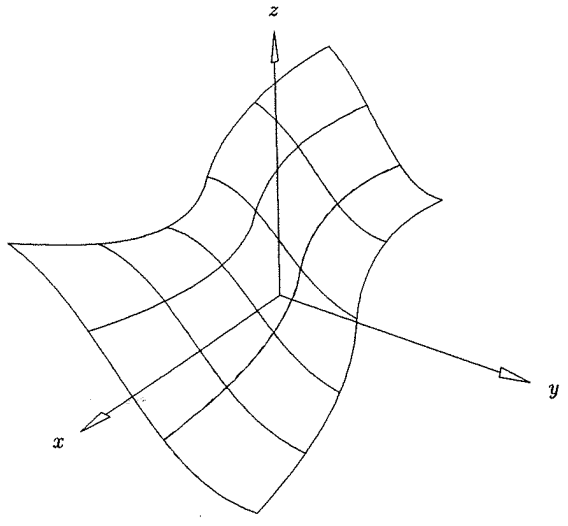


(b)

Figure 5.30. A (cubic × quadratic) surface for knot removal. (a) The control net; (b) the surface defined over $U = \{0, 0, 0, 0, 0.22, 0.3, 0.52, 0.7, 0.79, 1, 1, 1, 1\}$ and $V = \{0, 0, 0, 0.2, 0.42, 0.5, 0.81, 0.9, 1, 1, 1\}$.

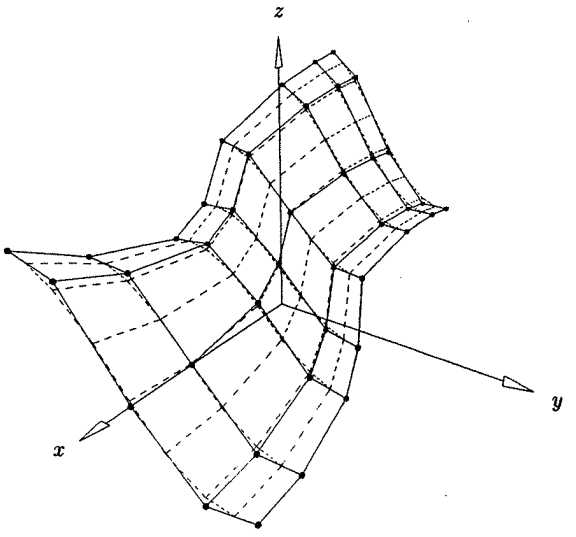


(a)

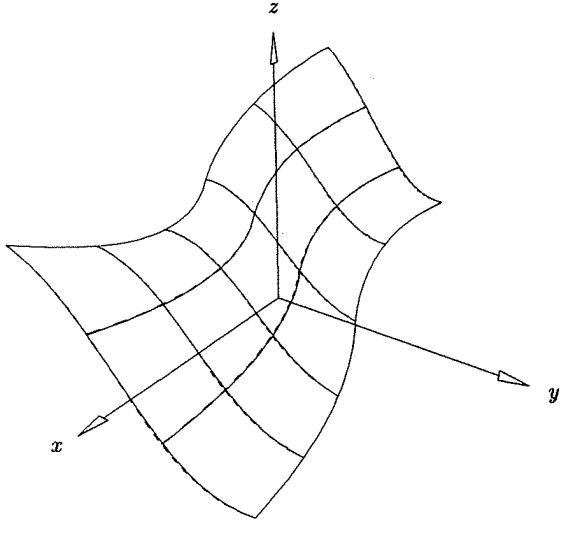


(b)

Figure 5.31. Remove all removable knots in both directions using the tolerance 0.05. (a) The control net of both the original (dashed) and the knot removed surface (solid); (b) isoparametric lines of both surfaces.

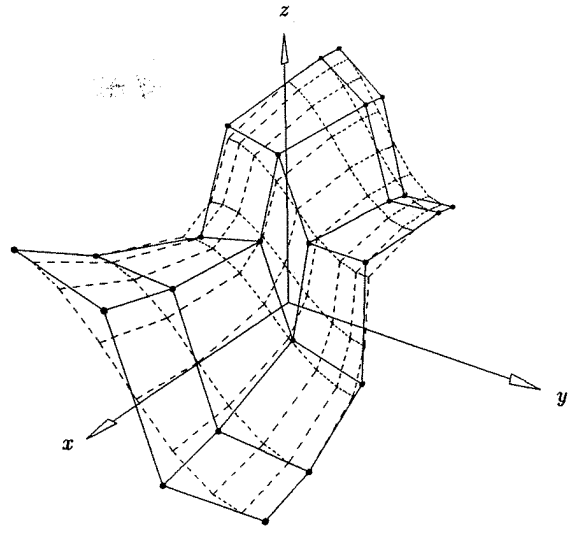


(a)

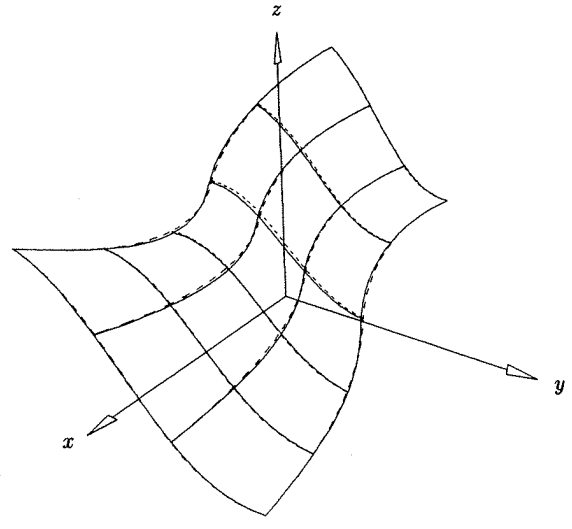


(b)

Figure 5.32. Remove all removable knots in both directions using the tolerance 0.1. (a) The control net of both the original (dashed) and the knot removed surface (solid); (b) isoparametric lines of both surfaces.

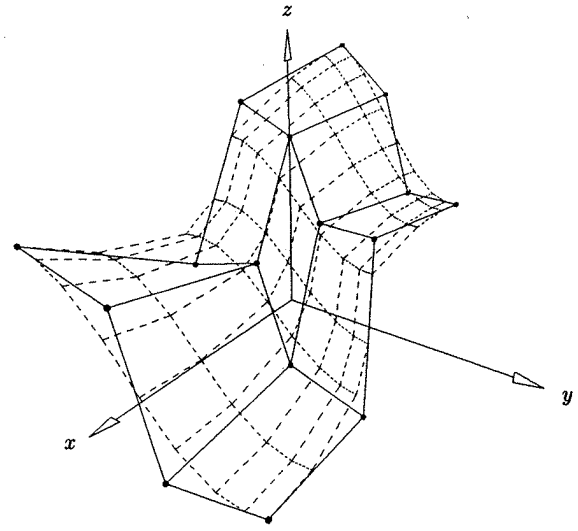


(a)

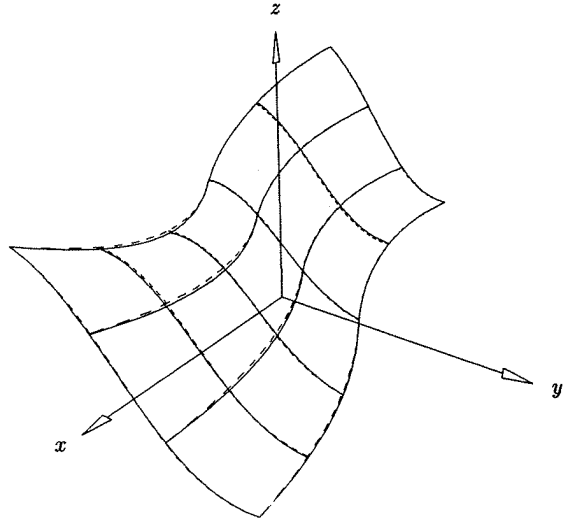


(b)

Figure 5.33. Remove all removable knots in both directions using the tolerance 0.3. (a) The control net of both the original (dashed) and the knot removed surface (solid); (b) isoparametric lines of both surfaces.

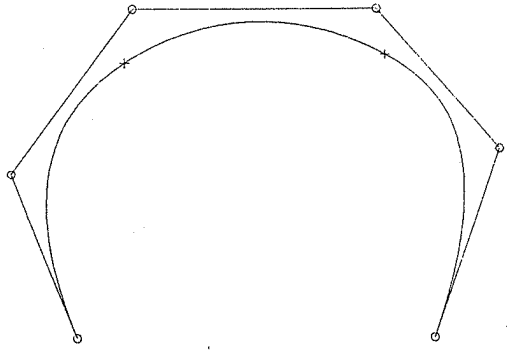


(a)

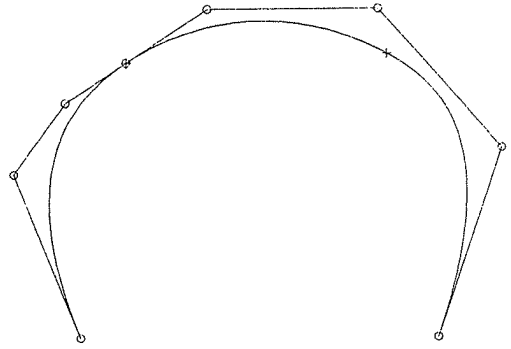


(b)

Figure 5.34. Remove all removable knots in both directions using the tolerance 0.5. (a) The control net of both the original (dashed) and the knot removed surface (solid); (b) isoparametric lines of both surfaces.



(a)



(b)

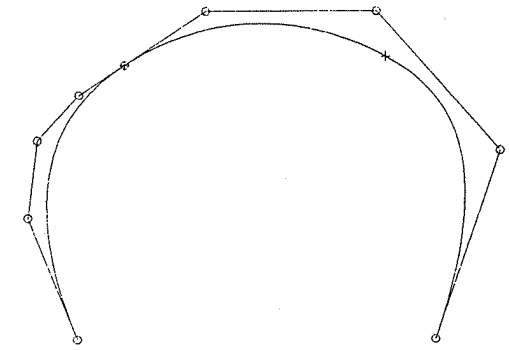
Figure 5.35. Degree elevation of a cubic curve defined over $\{0, 0, 0, 0, 3/10, 7/10, 1, 1, 1, 1\}$. (a) The original curve; (b) the first Bézier segment obtained via knot insertion.

The only remaining problem is to compute the $\{\mathbf{Q}_i\}$. An obvious but very inefficient method to do this is to solve a system of linear equations. Setting

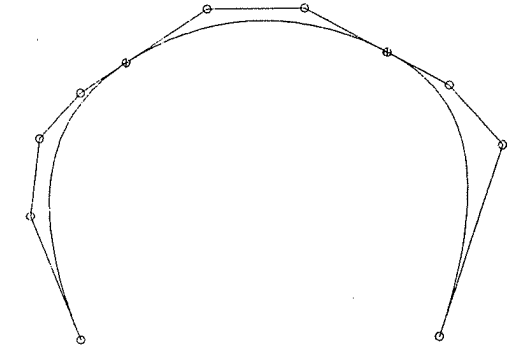
$$\sum_{i=0}^{\hat{n}} N_{i,p+1}(u)\,\mathbf{Q}_i = \sum_{i=0}^{n} N_{i,p}(u)\,\mathbf{P}_i$$

and evaluating the $N_{i,p}(u)$ and $N_{i,p+1}(u)$ at appropriate $\hat{n}+1$ values yields a banded system of $\hat{n}+1$ linear equations in the unknowns, $\mathbf{Q}_i$.

More efficient but mathematically more complicated methods are given by Prautzsch [Prau84], by Cohen et al. [Cohe85], and by Prautzsch and Piper [Prau91]. The algorithm due to Prautzsch and Piper is the most efficient for the general case, but Cohen et al. also give simple and efficient algorithms for low-degree special cases, such as linear to quadratic, and quadratic to cubic. All these algorithms raise the degree by 1 ($p \rightarrow p+1$).



(c)



(d)

Figure 5.35. *Continued.* (c) the first Bézier segment is degree elevated, (d) the second Bézier segment is computed.

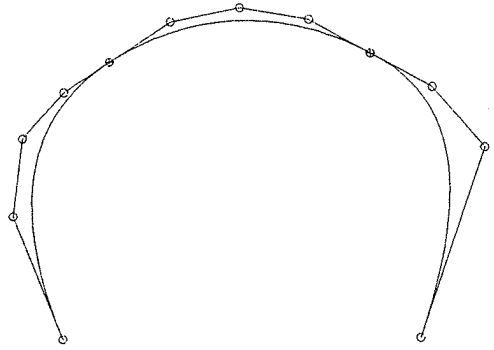
We present another algorithm here which is mathematically simpler, and competitive with that given in [Prau91], particularly in the case where the degree is to be raised by more than 1 (see [Pieg94]). The solution consists of applying three steps, on-the-fly, while moving through the knot vector. The steps are:

1. using the logic of Algorithm `A5.6`, extract the ith Bézier segment from the curve;
2. degree elevate the ith Bézier segment;
3. remove unnecessary knots separating the $(i-1)$th and ith segments.

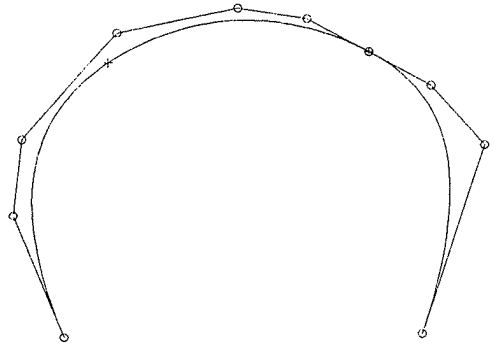
Figures 5.35a–5.35h show the progress of the algorithm. The original curve is a cubic with knot vector $U = \{0, 0, 0, 0, u_a, u_b, 1, 1, 1, 1\}$. It is elevated to fourth-degree. The figures are explained as:

a. the original curve;

b. u_a is inserted twice, bringing it to multiplicity 3;



(e)



(f)

Figure 5.35. *Continued.* (e) the second Bézier segment is degree elevated; (f) two occurrences of the knot $u = {}^3/_{10}$ are removed.

c. The Bézier segment on $[0, u_a]$ is degree elevated; this replaces the first four control points with five new ones (see subsequent text);

d. u_b is inserted twice to obtain the Bézier segment on $[u_a, u_b]$;

e. the Bézier segment on $[u_a, u_b]$ is degree elevated;

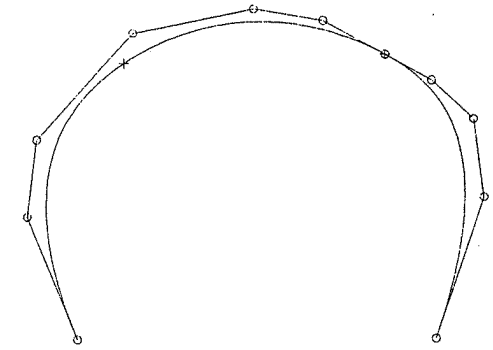
f. two occurrences of the knot u_a are removed;

g. the last segment $[u_b, 1]$ is degree elevated;

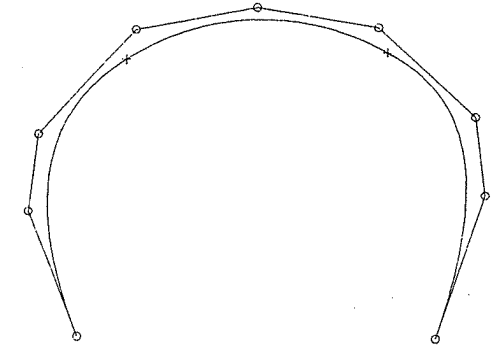
h. two occurrences of u_b are removed, yielding the final fourth-degree, C^2 continuous curve.

Step 1 is a straightforward application of Algorithm `A5.6`. It is modified to not return the Bézier segments, but rather to process them on-the-fly.

Step 2 degree elevates a Bézier segment. We present two formulas to do this. The first is the well-known formula (due to Forrest [Forr72]) to elevate from degree p to degree $p+1$. Let



(g)



(h)

Figure 5.35. *Continued.* (g) the third Bézier segment is degree elevated; (h) two occurrences of the knot $u = {}^7/_{10}$ are removed.

$$\mathbf{C}_p(u) = \sum_{i=0}^{p} B_{i,p}(u)\,\mathbf{P}_i$$

be a pth-degree Bézier curve. Its representation as a $(p+1)$th-degree curve is

$$\mathbf{C}_{p+1}(u) = \sum_{i=0}^{p+1} B_{i,p+1}(u)\,\mathbf{Q}_i \tag{5.34}$$

Setting these equal and multiplying $\mathbf{C}_p(u)$ by $\big(u + (1-u)\big)\ (= 1)$ yields

$$\sum_{i=0}^{p+1} B_{i,p+1}\,\mathbf{Q}_i = \big(u + (1-u)\big)\sum_{i=0}^{p} B_{i,p}\,\mathbf{P}_i$$

$$= \sum_{i=0}^{p} \big((1-u)\,B_{i,p} + uB_{i,p}\big)\,\mathbf{P}_i$$

Using the notation

$$\frac{p!}{i!(p-i)!} = \binom{p}{i}$$

and applying Eq. (1.8), we obtain

$$\sum_{i=0}^{p+1}\binom{p+1}{i}u^i(1-u)^{p+1-i}\mathbf{Q}_i$$

$$= \sum_{i=0}^{p}\binom{p}{i}\left(u^i(1-u)^{p+1-i} + u^{i+1}(1-u)^{p-i}\right)\mathbf{P}_i$$

$$= \sum_{i=0}^{p}\binom{p}{i}u^i(1-u)^{p+1-i}\mathbf{P}_i + \sum_{i=1}^{p+1}\binom{p}{i-1}u^i(1-u)^{p+1-i}\mathbf{P}_{i-1}$$

Equating coefficients of $u^i(1-u)^{p+1-i}$ yields

$$\binom{p+1}{i}\mathbf{Q}_i = \binom{p}{i}\mathbf{P}_i + \binom{p}{i-1}\mathbf{P}_{i-1}$$

From

$$\binom{p}{i}\Big/\binom{p+1}{i} = \frac{p!i!(p+1-i)!}{i!(p-i)!(p+1)!} = \frac{p+1-i}{p+1} = 1 - \frac{i}{p+1}$$

and

$$\binom{p}{i-1}\Big/\binom{p+1}{i} = \frac{p!i!(p+1-i)!}{(i-1)!(p+1-i)!(p+1)!} = \frac{i}{p+1}$$

it follows that

$$\mathbf{Q}_i = (1-\alpha_i)\mathbf{P}_i + \alpha_i\mathbf{P}_{i-1} \tag{5.35}$$

where

$$\alpha_i = \frac{i}{p+1} \qquad i = 0,\ldots,p+1$$

Notice that Eq. (5.35) represents a corner cutting process (see Figures 5.35c, 5.35e, and 5.35g). Regarding Eq. (5.35):

- it can be applied recursively to elevate the degree t times ($t \geq 1$);
- its growth rate is $O(t^2)$; if $t > 1$, recursive application of Eq. (5.35) involves some redundant computations;
- the α_is depend only on the degree, not on the particular Bézier segment to which they are applied, thus they can be computed one time and stored in a local array before processing of the segments begins;
- it is a convex combination scheme.

It is possible to degree elevate from p to $p+t$ in one step. Writing out several recursive applications of Eq. (5.35), and cleverly rearranging coefficients, yields

$$\mathbf{P}_i^t = \sum_{j=\max(0,i-t)}^{\min(p,i)} \frac{\binom{p}{j}\binom{t}{i-j}\mathbf{P}_j}{\binom{p+t}{i}} \qquad i = 0,\ldots,p+t \tag{5.36}$$

where $\mathbf{P}_i^t$ denotes the degree elevated control points after t-degree elevations. As an example, consider the case of $p = 2$ and $t = 4$, that is

$$\mathbf{P}_0^4 = \mathbf{P}_0$$

$$\mathbf{P}_1^4 = \frac{4}{6}\mathbf{P}_0 + \frac{2}{6}\mathbf{P}_1$$

$$\mathbf{P}_2^4 = \frac{6}{15}\mathbf{P}_0 + \frac{8}{15}\mathbf{P}_1 + \frac{1}{15}\mathbf{P}_2$$

$$\mathbf{P}_3^4 = \frac{4}{20}\mathbf{P}_0 + \frac{12}{20}\mathbf{P}_1 + \frac{4}{20}\mathbf{P}_2$$

$$\mathbf{P}_4^4 = \frac{1}{15}\mathbf{P}_0 + \frac{8}{15}\mathbf{P}_1 + \frac{6}{15}\mathbf{P}_2$$

$$\mathbf{P}_5^4 = \frac{2}{6}\mathbf{P}_1 + \frac{4}{6}\mathbf{P}_2$$

$$\mathbf{P}_6^4 = \mathbf{P}_2$$

As regards Eq. (5.36):

- its growth rate is $O(t)$, as the width of the scheme is fixed (bounded by $p+1$);
- the coefficients for each $\mathbf{P}_i^t$ sum to one (see example), hence it is a convex combination scheme;
- there are no redundant computations;
- the coefficient matrix is symmetric with respect to the row $^{p+t}/_2$;
- the coefficients depend both on p and t, but not on the particular Bézier segment to which they are applied; hence, they can be computed one time and stored in a local array before processing of the segments begins.

The knot removal of Step 3 is much less expensive than general knot removal as described in the previous section. There are two reasons for this:

- we know how many knots are removable – the number that were inserted;
- the knot vector has a specific structure in the neighborhood of the knot being removed.

As an example of this, Figure 5.36 shows a curve whose degree is being raised from 4 to 5 ($p = 4$). Let u_c, u_d, u_e be the three parameter values defining the two segments shown. Assuming the original multiplicity of u_d is 1, then three knot insertions are required to form the two Bézier segments (note that $\mathbf{C}(u_d) = \mathbf{P}_k^0$), and three removals are now necessary. However, we can skip the first two and compute $\mathbf{P}_{k-1}^3$ and $\mathbf{P}_k^3$ directly. This requires computing only two points in the form of Eq. (5.29), and no checking for equality of points is necessary. This compares to the six point computations and the three point equality checks required in the general case. Locally the knot vector has the form

$$\ldots, u_c, u_c, \underbrace{u_d, \ldots, u_d}_{p+1}, \underbrace{u_e, \ldots, u_e}_{p+1}, \ldots$$

where u_c has multiplicity of at least 2.

Algorithm A5.9 raises the degree from p to $p + t$, $t \geq 1$, by computing $\hat{n}$, $\hat{U}$, and the new $\mathbf{Q}_i$ (nh, Uh, and Qw); it uses Eq. (5.36) to degree elevate the Bézier segments. Min() and Max() compute the minimum and the maximum of two integers, respectively, and Bin(i,j) computes $\binom{i}{j}$, the binomial coefficient, which may be precomputed and stored for further efficiency. Let p be the degree of the original curve. The required local arrays are:

bezalfs[p+t+1][p+1]	: coefficients for degree elevating the Bézier segments;
bpts[p+1]	: pth-degree Bézier control points of the current segment;
ebpts[p+t+1]	: $(p+t)$th-degree Bézier control points of the current segment;
Nextbpts[p-1]	: leftmost control points of the next Bézier segment;
alphas[p-1]	: knot insertion αs.

```
ALGORITHM A5.9
DegreeElevateCurve(n,p,U,Pw,t,nh,Uh,Qw)
  {  /*  Degree elevate a curve t times.  */
     /*  Input:  n,p,U,Pw,t  */
     /*  Output: nh,Uh,Qw  */
   m = n+p+1;
   ph = p+t;     ph2 = ph/2;
    /* Compute Bézier degree elevation coefficients */
   bezalfs[0][0] = bezalfs[ph][p] = 1.0;
   for (i=1; i<=ph2; i++)
     {
     inv = 1.0/Bin(ph,i);
     mpi = Min(p,i);
```

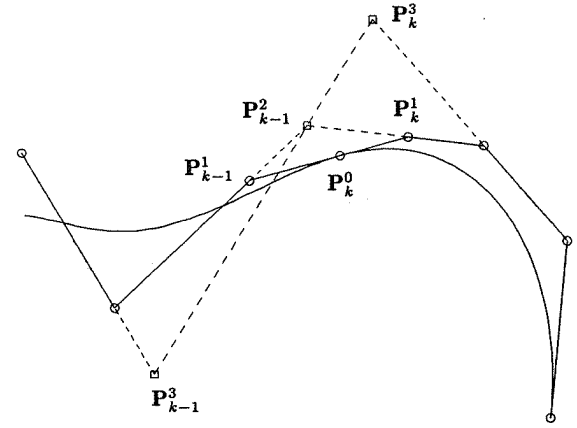



Figure 5.36. Removal of three knots from a quartic curve.

```
     for (j=Max(0,i-t); j<=mpi; j++)
       bezalfs[i][j] = inv*Bin(p,j)*Bin(t,i-j);
     }
   for (i=ph2+1; i<=ph-1; i++)
     {
     mpi = Min(p,i);
     for (j=Max(0,i-t); j<=mpi; j++)
       bezalfs[i][j] = bezalfs[ph-i][p-j];
     }
   mh = ph;   kind = ph+1;
   r = -1;   a = p;
   b = p+1;   cind = 1;
   ua = U[0];
   Qw[0] = Pw[0];
   for (i=0; i<=ph; i++)   Uh[i] = ua;
     /* Initialize first Bézier seg */
   for (i=0; i<=p; i++)   bpts[i] = Pw[i];
   while (b < m)   /* Big loop thru knot vector */
     {
     i = b;
     while (b < m && U[b] == U[b+1])   b = b+1;
     mul = b-i+1;
     mh = mh+mul+t;
     ub = U[b];
     oldr = r;     r = p-mul;
        /* Insert knot u(b) r times */
     if (oldr > 0)   lbz = (oldr+2)/2;   else   lbz = 1;
```

```
if (r > 0)   rbz = ph-(r+1)/2;   else   rbz = ph;
if (r > 0)
  {  /* Insert knot to get Bézier segment */
  numer = ub-ua;
  for (k=p; k>mul; k--)
    alfs[k-mul-1] = numer/(U[a+k]-ua);
  for (j=1; j<=r; j++)
    {
    save = r-j;   s = mul+j;
    for (k=p; k>=s; k--)
      {
      bpts[k] = alfs[k-s]*bpts[k] +
                (1.0-alfs[k-s])*bpts[k-1];
      }
    Nextbpts[save] = bpts[p];
    }
  }  /* End of "insert knot" */
for (i=lbz; i<=ph; i++)   /* Degree elevate Bezier */
  {  /* Only points lbz,...,ph are used below */
  ebpts[i] = 0.0;
  mpi = Min(p,i);
  for (j=Max(0,i-t); j<=mpi; j++)
    ebpts[i] = ebpts[i] + bezalfs[i][j]*bpts[j];
  }  /* End of degree elevating Bezier */
if (oldr > 1)
  {  /* Must remove knot u=U[a] oldr times */
  first = kind-2;   last = kind;
  den = ub-ua;
  bet = (ub-Uh[kind-1])/den;
  for (tr=1; tr<oldr; tr++)
    {  /* Knot removal loop */
    i = first;    j = last;    kj = j-kind+1;
    while (j-i > tr)  /* Loop and compute the new */
      {   /* control points for one removal step */
      if (i < cind)
        {
        alf = (ub-Uh[i])/(ua-Uh[i]);
        Qw[i] = alf*Qw[i] + (1.0-alf)*Qw[i-1];
        }
      if( j >= lbz)
        {
        if( j-tr <= kind-ph+oldr )
          {
          gam = (ub-Uh[j-tr])/den;
          ebpts[kj] = gam*ebpts[kj]+(1.0-gam)*ebpts[kj+1];
```

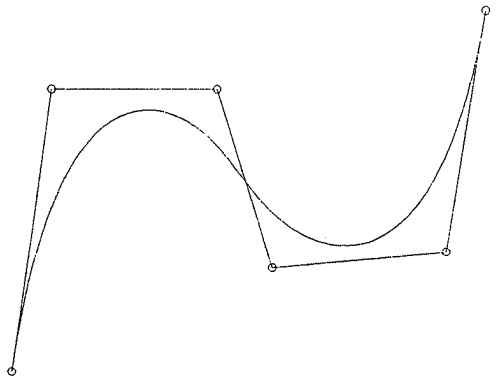


```
          {
          else
          {
          ebpts[kj] = bet*ebpts[kj]+(1.0-bet)*ebpts[kj+1];
          {
        }
      i = i+1;   j = j-1;   kj = kj-1;
      }
    first = first-1;    last = last+1;
    }
  }  /* End of removing knot, u=U[a] */
if (a != p)  /* Load the knot ua */
  for (i=0; i<ph-oldr; i++)
    {  Uh[kind] = ua;   kind = kind+1;  }
for (j=lbz; j<=rbz; j++)  /* Load ctrl pts into Qw */
  {  Qw[cind] = ebpts[j];   cind = cind+1;  }
if (b < m)
  {  /* Set up for next pass thru loop */
  for (j=0; j<r; j++)    bpts[j] = Nextbpts[j];
  for (j=r; j<=p; j++)   bpts[j] = Pw[b-p+j];
  a = b;   b = b+1;   ua = ub;
  }
  else
      /* End knot */
    for (i=0; i<=ph; i++)  Uh[kind+i] = ub;
}  /* End of while-loop (b < m) */
nh = mh-ph-1;
}
```

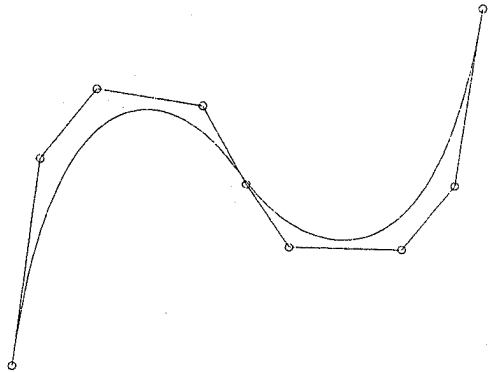
Several curve degree elevation examples are shown in Figures 5.37a–5.37d. The original third-degree curve (Figure 5.37a) is raised to fourth-, fifth-, and seventh-degrees in Figures 5.37b, 5.37c, and 5.37d, respectively. Note that the control polygon converges to the curve as the degree is raised.

Let $\mathbf{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\mathbf{P}^w_{i,j}$ be a NURBS surface. Degree elevation is accomplished for surfaces by applying it to the rows/columns of control points. In particular, we elevate the degree p (u direction) by applying Algorithm A5.9 to each of the $m+1$ columns of control points. The v direction degree q is elevated by applying Algorithm A5.9 to each of the $n+1$ rows of control points. An efficient organization of a surface degree elevation algorithm which requires storage of Bézier strips is Algorithm A5.10.

```
ALGORITHM A5.10
DegreeElevateSurface(n,p,U,m,q,V,Pw,dir,t,nh,Uh,mh,Vh,Qw)
  {  /*  Degree elevate a surface t times.  */
     /*  Input:  n,p,U,m,q,V,Pw,dir,t  */
     /*  Output: nh,Uh,mh,Vh,Qw  */
```



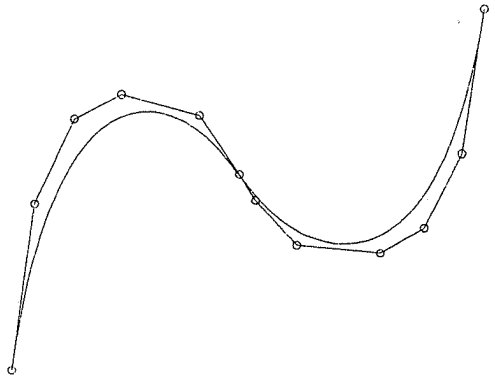
(a)



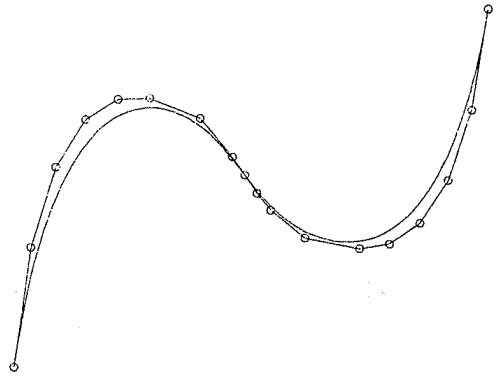
(b)

Figure 5.37. Curve degree elevation example. (a) The original cubic curve defined over $\{0, 0, 0, 0, 4/10, 7/10, 1, 1, 1, 1\}$; (b) the degree is elevated by one; (c) the degree is elevated by two; (d) the degree is elevated by four.

```
if (dir == U_DIRECTION)
  {
  allocate memory for Bezier and NextBezier strips;
  initialize knot vectors and first row of control points;
  initialize Bezier strip;
  set variables;
  while(b<m)
    {
    get multiplicity;
    get ub, r, oldr, etc;
    save alfas
```

(c)



(d)

Figure 5.37. (*Continued.*)

```
    for (j=0; j<=m; j++)
      {
      insert knot;
      degree elevate Bezier row[j];
      remove knot;
      save new control points;
      initialize for next pass through;
      }
    update knot vector;
    update variables;
    get end knots;
    }
}
```

```
if (dir == V_DIRECTION)
  {
  /* Similar code as above with u- and v-directional
    parameters switched */
  }
}
```

Surface degree elevation examples are shown in Figures 5.38a–5.38e. The original $(3,2)$th-degree surface (Figure 5.38a) is raised to $(4,3)$, $(5,4)$, $(6,6)$, and $(9,9)$th-degrees in Figures 5.38b–5.38e, respectively; the original net is shown dashed, and the degree elevated net is drawn solid.

5.6 Degree Reduction

Let $\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{Q}_i$ be a pth-degree B-spline curve on the knot vector

$$U = \{\underbrace{a,\ldots,a}_{p+1}, \underbrace{u_1,\ldots,u_1}_{m_1}, \ldots, \underbrace{u_s,\ldots,u_s}_{m_s}, \underbrace{b,\ldots,b}_{p+1}\}$$

As usual, $\mathbf{C}(u)$ can be rational or nonrational; we drop the w superscript for the remainder of this section. $\mathbf{C}(u)$ is *degree reducible* if it has a precise representation of the form

$$\mathbf{C}(u) = \hat{\mathbf{C}}(u) = \sum_{i=0}^{\hat{n}} N_{i,p-1}(u)\mathbf{P}_i \tag{5.37}$$

on the knot vector

$$\hat{U} = \{\underbrace{a,\ldots,a}_{p}, \underbrace{u_1,\ldots,u_1}_{m_1-1}, \ldots, \underbrace{u_s,\ldots,u_s}_{m_s-1}, \underbrace{b,\ldots,b}_{p}\} \tag{5.38}$$

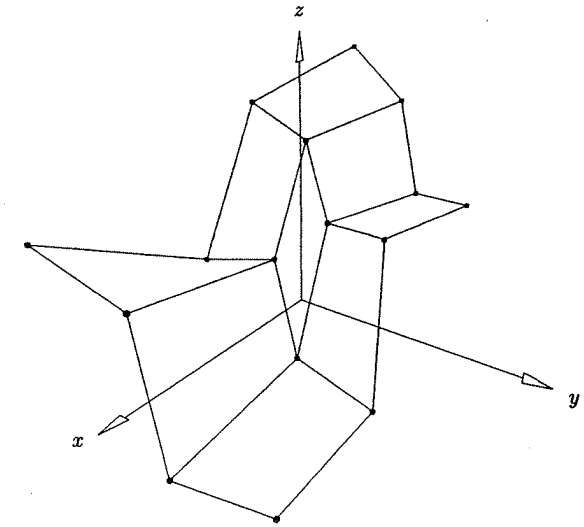
Clearly

$$\hat{n} = n - s - 1 \tag{5.39}$$

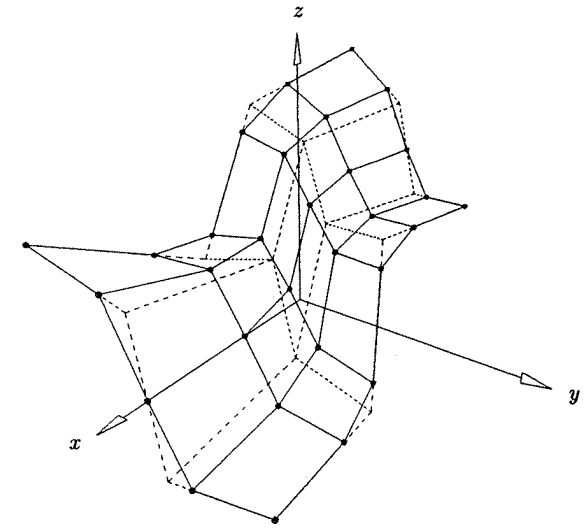
Note that m_i can be 1, which implies that the knot u_i is not present in $\hat{U}$. This means that u_i was precisely removable from $\mathbf{C}(u)$. Although it is always possible to degree elevate a curve, clearly a curve may not be degree reducible. The situation is similar to knot removal in that the problem is over-determined, that is, any algorithm for degree reduction must produce more equations than unknown $\mathbf{P}_i$. Due to floating point round-off error, one can never expect $\hat{\mathbf{C}}(u)$ to coincide precisely with $\mathbf{C}(u)$, and therefore our algorithm must measure the error $E(u) = \mid \mathbf{C}(u) - \hat{\mathbf{C}}(u) \mid$ and declare $\mathbf{C}(u)$ to be degree reducible only if

$$\max_u E(u) \leq \texttt{TOL}$$

for some user-specified tolerance, TOL. We consider only *precise* degree reduction in most of this section, i.e., we assume that TOL is very small. The main application is to reverse the degree elevation process. For example, degree reduction should be applied to each constituent piece when decomposing a composite

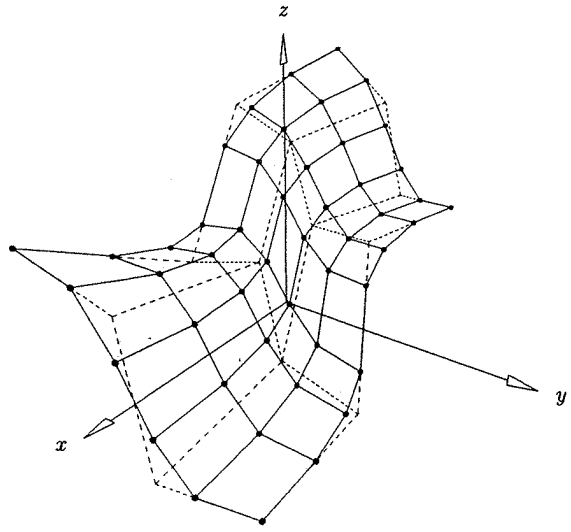


(a)

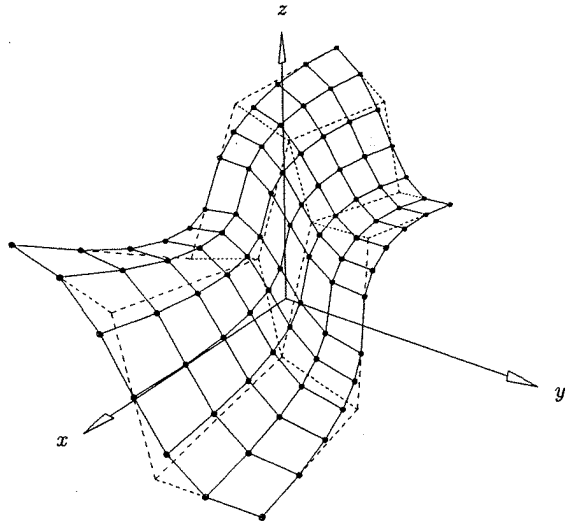


(b)

Figure 5.38. Surface degree elevation example. (a) The original (cubic × quadratic) surface net, the surface defined over $U = \{0,0,0,0,1,1,1,1\}$, and $V = \{0,0,0,\frac{1}{2},1,1,1\}$; (b) the degree elevated by one in both directions; (c) the degree elevated by two in both directions; (d) the degree elevated by three in the u direction, by four in the v direction; (e) the degree elevated by six in the u direction, by seven in the v direction.



(c)

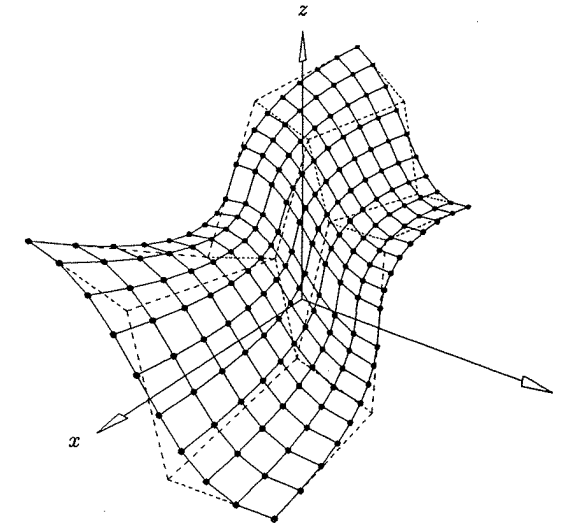


(d)

Figure 5.38. (*Continued.*)

curve, because degree elevation may have been required in the composition process. The material in this section is taken from [Pieg95].

Degree reduction of Bézier curves is relatively well understood, and there exist a number of algorithms (e.g., see references [Forr72; Dann85; Lach88; Watk88;



(e)

Figure 5.38. (*Continued.*)

Wein92; Eck93]). Following the strategy developed in the previous section for degree elevation, we present a three-step algorithm for degree reduction of B-spline curves:

repeat
  extract the ith Bézier segment from the B-spline curve,
  degree reduce the ith Bézier piece,
  remove unnecessary knots between the $(i-1)$th and the
    ith segments,
until done

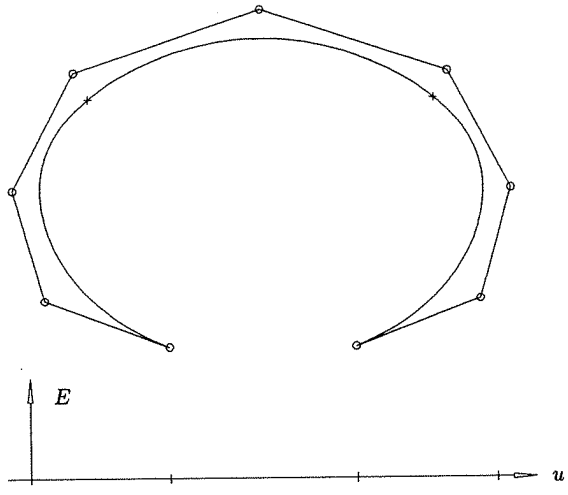
Error is produced in both the Bézier degree reduction and the knot removal steps, and our algorithm accumulates both types of error and exits immediately if TOL is exceeded on any knot span. Figures 5.39a–5.39h show walk-through examples explaining how the algorithm works. The details of each step are:

a. original fourth-degree B-spline curve defined over the knot vector

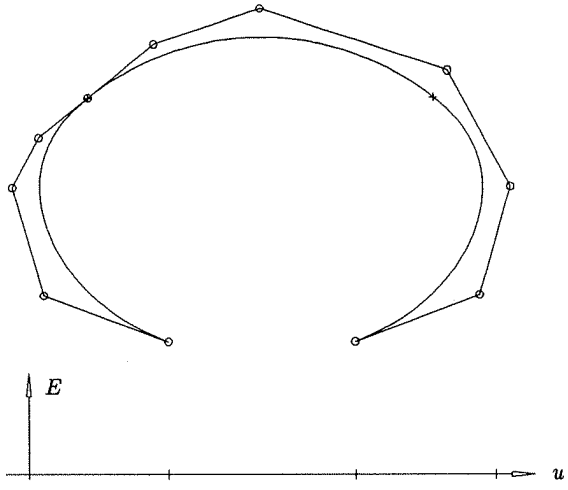
$$U = \{0,0,0,0,0,u_1,u_1,u_2,u_2,1,1,1,1,1\}$$

The coordinate system (u, E) is used to graph the error over each knot span as a function of u as the algorithm sweeps out the entire knot vector;

b. the knot u_1 is inserted twice, bringing the total multiplicity to four; the first Bézier piece is obtained;



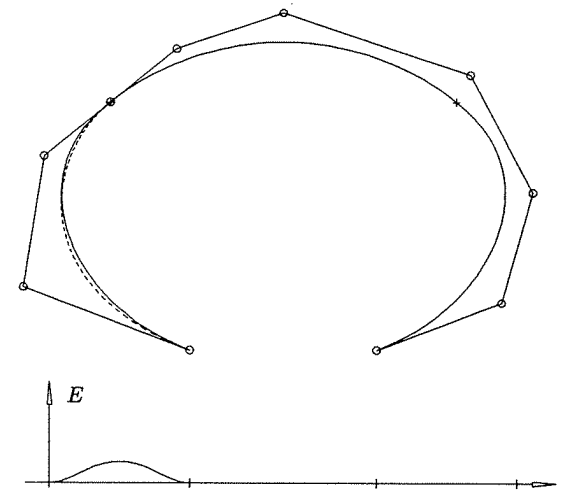
(a)



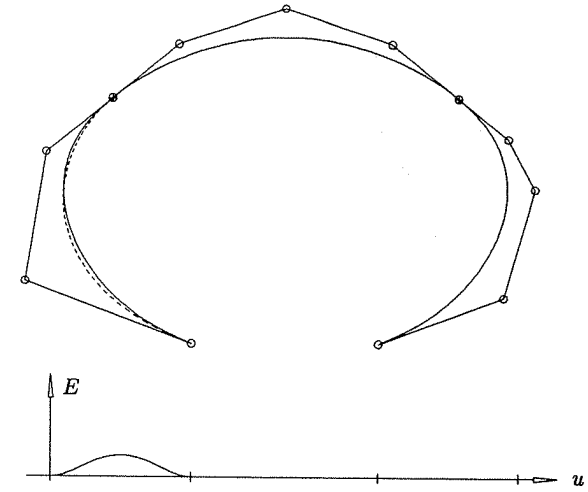
(b)

Figure 5.39. Degree reduction of a B-spline curve from fourth-degree to third-degree. (a) Fourth-degree B-spline curve to be degree reduced; (b) first Bézier segment is extracted.

c. the first Bézier segment is degree reduced, which replaces the first five control points by four new ones; this is the first time error is introduced as graphed in the (u, E) system. The solid curve is the original curve, whereas the dashed one is the approximation;



(c)



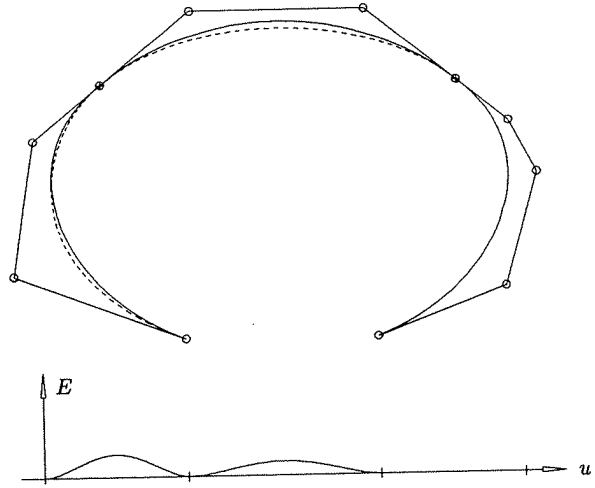
(d)

Figure 5.39. *Continued.* (c) first Bézier segment is degree reduced; (d) second Bézier segment is extracted.

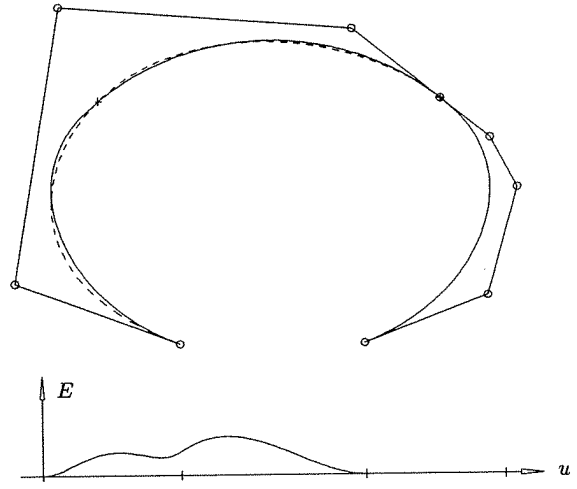
d. the knot u_2 is inserted twice to obtain the second Bézier segment;

e. the second Bézier segment is degree reduced. Now there is error over two segments; both are Bézier degree reduction errors;

f. two occurrences of the knot u_1 are removed; notice how knot removal introduces additional error that affects more than one knot span;



(e)



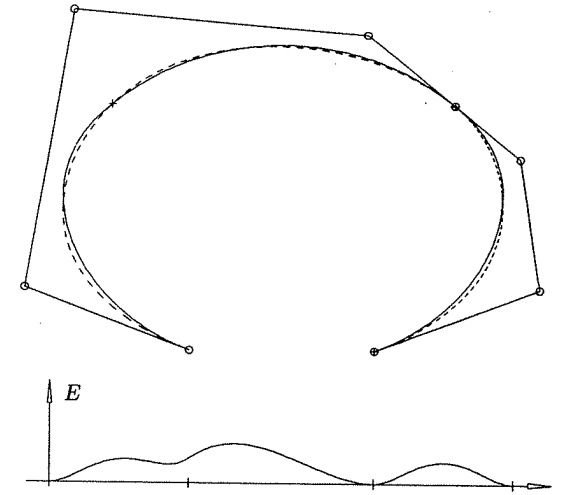
(f)

Figure 5.39. *Continued.* (e) second Bézier segment is degree reduced; (f) first interior multiple knot is removed twice.

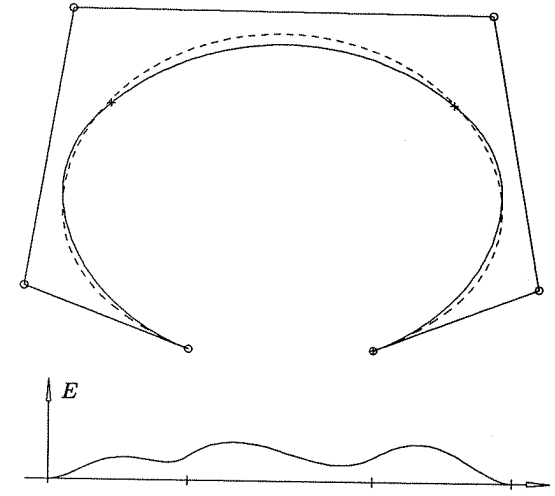
g. the last Bézier segment is degree reduced;

h. two occurrences of u_2 are removed, yielding the final curve.

Curve decomposition, knot removal, and bounding the knot removal error were covered in Sections 5.3 and 5.4.



(g)



(h)

Figure 5.39. *Continued.* (g) third Bézier segment is degree reduced; (h) second interior multiple knot is removed twice.

We focus now on Bézier degree reduction. Let

$$\mathbf{C}(u) = \sum_{i=0}^{p} B_{i,p}(u)\,\mathbf{Q}_i$$

be a pth degree Bézier curve, and denote its degree reduced counterpart by

$$\hat{\mathbf{C}}(u) = \sum_{i=0}^{p-1} B_{i,p-1}(u)\mathbf{P}_i$$

We compute the unknown $\mathbf{P}_i$ by turning around the Bézier degree elevation formula, Eqs. (5.34) and (5.35); let

$$r = \frac{p-1}{2} \tag{5.40}$$

(integer division) and denote the degree elevation coefficients by α_i, that is

$$\alpha_i = \frac{i}{p}$$

(we need coefficients for degree elevation from $p-1$ to p). There are two cases to consider, p is even, and p is odd. Assume first that p is even. Solving Eq. (5.35) for the $\mathbf{P}_i$, we obtain

$$\begin{aligned}
\mathbf{P}_0 &= \mathbf{Q}_0 \\
\mathbf{P}_i &= \frac{\mathbf{Q}_i - \alpha_i \mathbf{P}_{i-1}}{1-\alpha_i} \qquad i = 1, \ldots, r \\
\mathbf{P}_i &= \frac{\mathbf{Q}_{i+1} - (1-\alpha_{i+1})\mathbf{P}_{i+1}}{\alpha_{i+1}} \qquad i = p-2, \ldots, r+1 \\
\mathbf{P}_{p-1} &= \mathbf{Q}_p
\end{aligned} \tag{5.41}$$

Now assume p is odd; then the $\mathbf{P}_i$ are computed as

$$\begin{aligned}
\mathbf{P}_0 &= \mathbf{Q}_0 \\
\mathbf{P}_i &= \frac{\mathbf{Q}_i - \alpha_i \mathbf{P}_{i-1}}{1-\alpha_i} \qquad i = 1, \ldots, r-1 \\
\mathbf{P}_i &= \frac{\mathbf{Q}_{i+1} - (1-\alpha_{i+1})\mathbf{P}_{i+1}}{\alpha_{i+1}} \qquad i = p-2, \ldots, r+1 \\
\mathbf{P}_r &= \frac{1}{2}\left(\mathbf{P}_r^L + \mathbf{P}_r^R\right) \\
\mathbf{P}_{p-1} &= \mathbf{Q}_p
\end{aligned} \tag{5.42}$$

where

$$\mathbf{P}_r^L = \frac{\mathbf{Q}_r - \alpha_r \mathbf{P}_{r-1}}{1-\alpha_r}$$

$$\mathbf{P}_r^R = \frac{\mathbf{Q}_{r+1} - (1-\alpha_{r+1})\mathbf{P}_{r+1}}{\alpha_{r+1}}$$

Note that the odd case differs from the even case only in that, for symmetry reasons, the middle control point, $\mathbf{P}_r$, is computed as the average of two components computed from the left and right, respectively. In general, the resulting $(p-1)$th degree curve, $\hat{\mathbf{C}}(u)$, is an approximation to $\mathbf{C}(u)$; they coincide only when $\mathbf{C}(u)$ is precisely degree reducible. Piegl and Tiller [Pieg95] derive the error bounds for the approximation

p even:

$$\left|\mathbf{C}(u) - \hat{\mathbf{C}}(u)\right| = \left|B_{r+1,p}(u)\left[\mathbf{Q}_{r+1} - \frac{1}{2}(\mathbf{P}_r + \mathbf{P}_{r+1})\right]\right| \tag{5.43}$$

p odd:

$$\left|\mathbf{C}(u) - \hat{\mathbf{C}}(u)\right| = \frac{1}{2}(1-\alpha_r)\left|(B_{r,p}(u) - B_{r+1,p}(u))\left(\mathbf{P}_r^L - \mathbf{P}_r^R\right)\right| \tag{5.44}$$

Figures 5.40a and 5.40b show examples of reducing the degree of a Bézier curve from seven to six, and six to five, respectively. The error curves are graphed on the bottom of the figures. We remark that:

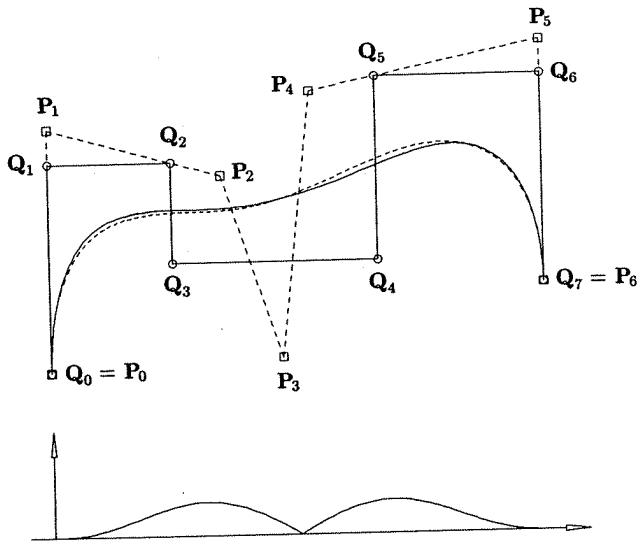
- Equations (5.43) and (5.44) express the *parametric* error (distance between points at corresponding parameter values); the maximums of geometric and parametric errors are not necessarily at the same u value;
- even if $\mathbf{C}(u)$ is not degree reducible, Eqs. (5.41) and (5.42) produce $p/2$ precise control points from the left and $p/2$ precise ones from the right, in the sense that if $\hat{\mathbf{C}}(u)$ is degree elevated to yield $\bar{\mathbf{C}}(u)$, then $\mathbf{C}(u)$ and $\bar{\mathbf{C}}(u)$ have the same first and last $p/2$ control points. Hence, $\mathbf{C}(u)$ and $\hat{\mathbf{C}}(u)$ have the same derivatives up to order $p/2-1$ at both ends. This is an important property from the standpoint of B-spline degree reduction, since it implies that up to $C^{p/2-1}$ continuity is maintained in the Bézier degree reduction steps, which in turn implies that the first $p/2-1$ knot removal steps produce no error;
- for p even the maximum error occurs at $u = 1/2$ (see Figure 5.40b); for p odd the error is zero at $u = 1/2$, and it has two peaks a bit to the left and right of $u = 1/2$ (Figure 5.40a). The odd case is analyzed in more detail elsewhere by Piegl and Tiller [Pieg95].

We now present an algorithm to degree reduce a B-spline curve, subject to a maximum error tolerance, `TOL`. If the curve is rational, then `TOL` should be adjusted as in Eq. (5.30). We maintain an error vector, e_i, $i = 0, \ldots, m-1$, which we use to accumulate error for each knot span. The e_i are initialized to zero. When the ith span is Bézier degree reduced, the incurred error is added to e_i. For simplicity we drop the scalar functions (which are bounded by 1) from Eqs. (5.43) and (5.44) and use maximum error bounds

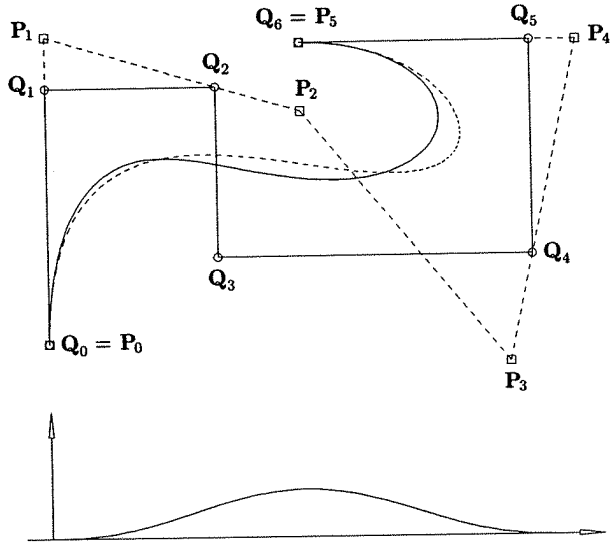
$$\left|\mathbf{C}(u) - \hat{\mathbf{C}}(u)\right| \le \left|\mathbf{Q}_{r+1} - \frac{1}{2}\left(\mathbf{P}_r + \mathbf{P}_{r+1}\right)\right| \tag{5.45}$$

and

$$\left|\mathbf{C}(u) - \hat{\mathbf{C}}(u)\right| \le \left|\mathbf{P}_r^L - \mathbf{P}_r^R\right| \tag{5.46}$$



(a)



(b)

Figure 5.40. Bézier degree reduction. (a) From degree seven to degree six; (b) from degree six to degree five.

The algorithm in [Pieg95] computes tighter error bounds for both the Bézier degree reduction and knot removal steps. When removing a knot, the maximum knot removal error is added to each e_i whose span is affected by the removal. The local arrays used are:

`bpts[p+1]`	:	Bézier control points of the current segment;
`Nextbpts[p-1]`	:	leftmost control points of the next Bézier segment;
`rbpts[p]`	:	degree reduced Bézier control points;
`alphas[p-1]`	:	knot insertion alphas;
`e[m]`	:	error vector.

A supporting routine, `BezDegreeReduce(bpts,rbpts,MaxErr)`, which implements Bézier degree reduction and computation of the maximum error, is used. This routine uses Eqs. (5.41), (5.42), (5.45), and (5.46). A return code of 1 indicates the curve was not degree reducible; if the curve is reducible, $\hat{n}$, $\hat{U}$, and the $\mathbf{P}_i$ are computed (`nh, Uh, Pw`) and a 0 code is returned.

```
ALGORITHM A5.11
DegreeReduceCurve(n,p,U,Qw,nh,Uh,Pw)
  { /*  Degree reduce a curve from p to p-1.  */
    /*  Input:  n,p,U,Qw  */
    /*  Output: nh,Uh,Pw  */
  ph = p-1;   mh = ph;   /* Initialize some variables */
  kind = ph+1;   r = -1;   a = p;
  b = p+1;   cind = 1;   mult = p;
  m = n+p+1;   Pw[0] = Qw[0];
  for (i=0; i<=ph; i++) /* Compute left end of knot vector */
    Uh[i] = U[0];
  for (i=0; i<=p; i++)  /* Initialize first Bézier segment */
    bpts[i] = Qw[i];
  for (i=0; i<m; i++)   /* Initialize error vector */
    e[i] = 0.0;
     /* Loop through the knot vector */
  while (b < m)
    { /* First compute knot multiplicity */
    i = b;
    while (b < m && U[b] == U[b+1])   b = b+1;
    mult = b-i+1;   mh = mh+mult-1;
    oldr = r;   r = p-mult;
    if (oldr > 0)  lbz = (oldr+2)/2;  else  lbz = 1;
       /* Insert knot U[b] r times */
    if (r > 0)
      {
      numer = U[b]-U[a];
      for (k=p; k>=mult; k--)
```

```
    alphas[k-mult-1] = numer/(U[a+k]-U[a]);
  for (j=1; j<=r; j++)
    {
    save = r-j;    s = mult+j;
    for (k=p; k>=s; k--)
      bpts[k] = alphas[k-s]*bpts[k]
                          + (1.0-alphas[k-s])*bpts[k-1];
    Nextbpts[save] = bpts[p];
    }
  }
    /* Degree reduce Bézier segment */
BezDegreeReduce(bpts,rbpts,MaxErr);
e[a] = e[a]+MaxErr;
if (e[a] > TOL)
  return(1);    /* Curve not degree reducible */
    /* Remove knot U[a] oldr times */
if (oldr > 0)
  {
  first = kind;    last = kind;
  for (k=0; k<oldr; k++)
    {
    i = first;    j = last;    kj = j-kind;
    while (j-i > k)
      {
      alfa = (U[a]-Uh[i-1])/(U[b]-Uh[i-1]);
      beta = (U[a]-Uh[j-k-1])/(U[b]-Uh[j-k-1]);
      Pw[i-1] = (Pw[i-1]-(1.0-alfa)*Pw[i-2])/alfa;
      rbpts[kj] = (rbpts[kj]-beta*rbpts[kj+1])/(1.0-beta);
      i = i+1;    j = j-1;    kj = kj-1;
      }
       /* Compute knot removal error bounds (Br) */
    if (j-i < k)  Br = Distance4D(Pw[i-2],rbpts[kj+1]);
      else
      {
      delta = (U[a]-Uh[i-1])/(U[b]-Uh[i-1]);
      A = delta*rbpts[kj+1]+(1.0-delta)*Pw[i-2];
      Br = Distance4D(Pw[i-1],A);
      }
       /* Update the error vector */
    K = a+oldr-k;    q = (2*p-k+1)/2;
    L = K-q;
    for (ii=L; ii<=a; ii++)
      {  /* These knot spans were affected */
      e[ii] = e[ii] + Br;
      if (e[ii] > TOL)
```



```
        return(1);    /* Curve not degree reducible */
      }
    first = first-1;    last = last+1;
    }  /* End for (k=0; k<oldr; k++) loop */
  cind = i-1;
  }  /* End if (oldr > 0) */
    /* Load knot vector and control points */
  if (a != p)
    for (i=0; i<ph-oldr; i++)
      {  Uh[kind] = U[a];    kind = kind+1;  }
  for (i=lbz; i<=ph; i++)
    {  Pw[cind] = rbpts[i];    cind = cind+1;  }
    /* Set up for next pass through */
  if (b < m)
    {
    for (i=0; i<r; i++)    bpts[i] = Nextbpts[i];
    for (i=r; i<=p; i++)   bpts[i] = Qw[b-p+i];
    a = b;    b = b+1;
    }
  else
    for (i=0; i<=ph; i++)   Uh[kind+i] = U[b];
  }  /* End of while (b < m) loop */
nh = mh-ph-1;
return(0);
}
```

Figure 5.41 shows an example of reducing the degree from five to four. The original knot vector is

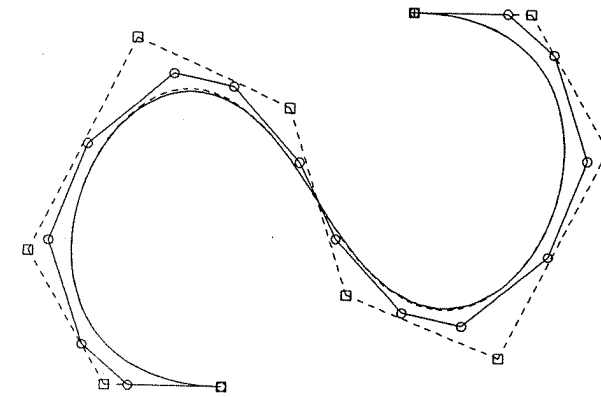
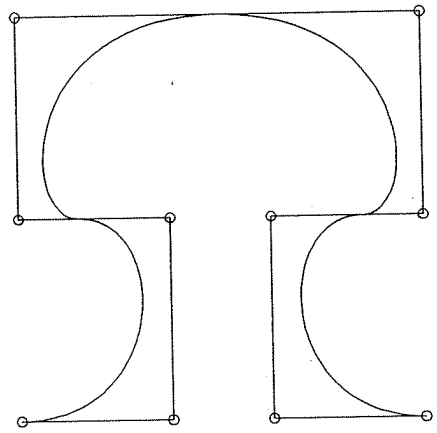


Figure 5.41. Degree reduction of a B-spline curve from degree five to four.

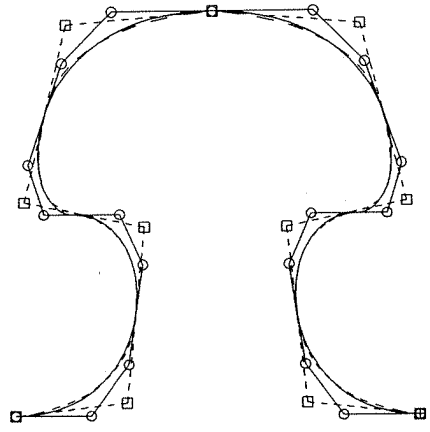
$$U = \{0,0,0,0,0,0,0.15,0.15,0.3,0.3,0.5,0.5,$$
$$0.7,0.7,0.85,0.85,1,1,1,1,1,1\}$$

The solid curve is the original curve, whereas the dashed one is the degree reduced curve. Figure 5.42a shows a cubic curve defined on the knot vector

$$U = \left\{0,0,0,0,\frac{3}{10},\frac{3}{10},\frac{1}{2},\frac{1}{2},\frac{7}{10},\frac{7}{10},1,1,1,1\right\}$$



(a)



(b)

Figure 5.42. A cubic curve. (a) The cubic curve is not degree reducible; (b) the curve is degree reducible after knot refinement.

It is not possible to approximate this curve with a second-degree B-spline curve to within a reasonable tolerance. After refining the knot vector to

$$U^{(r)} = \{0,0,0,0,0.15,0.15,0.3,0.3,0.4,0.4,0.5,0.5,0.5,$$
$$0.6,0.6,0.7,0.7,0.85,0.85,1,1,1,1\}$$

a reasonable approximation is possible, as shown in Figure 5.42b. Notice the discontinuity at $u = 1/2$. This is due to the triple knot in the original curve. The refinement brings out an interesting point: the quality of the approximation can be improved by introducing extra knots with at least multiplicity two. Hence, together with knot refinement, the tools of this section can be used to develop algorithms to approximate high-degree curves with lower-degree curves.

Finally, we remark that the u- or v-degree of a surface can be reduced by applying these curve techniques to the rows or columns of control points.

Exercises

5.1. Let $p = 2$ and $U = \{0,0,0,1,2,3,4,4,5,5,5\}$. $\mathbf{C}(u) = \sum_{i=0}^{7} N_{i,2}(u)\mathbf{P}_i$, a B-spline curve, is defined on U by the control points $\{(-6,-1),(-5,2),(-3,3),(-1,2),(0,0),(3,1),(3,3),(1,5)\}$. Using repeated knot insertion, compute $\mathbf{C}(5/4)$ and $\mathbf{C}'(5/4)$. Check that Eqs. (5.12) – (5.14) and (5.16) – (5.18) hold true for this example. Sketch the curve and its control polygon.

5.2. Use the same curve as in Exercise 5.1. What value of u must be inserted as a knot in order to cause the point $(-3/4, 3/2)$ to become a control point?

5.3. Consider the B-spline surface

$$\mathbf{S}(u,v) = \sum_{i=0}^{7}\sum_{j=0}^{8} N_{i,2}(u)N_{j,3}(v)\mathbf{P}_{i,j}$$

where $$U = \left\{0,0,0,\frac{1}{5},\frac{3}{10},\frac{3}{5},\frac{4}{5},\frac{9}{10},1,1,1\right\}$$

and $$V = \left\{0,0,0,0,\frac{1}{10},\frac{2}{5},\frac{1}{2},\frac{7}{10},\frac{4}{5},1,1,1,1\right\}$$

Suppose you want to modify the surface shape slightly in the region corresponding to the rectangular area of parameter space given by $3/10 < u < 6/10$ and $1/2 < v < 7/10$. You want to do this by adding knots (control points) until you have at least one control point which you can freely move, without changing the continuity of the surface with respect to u and v, and without changing the surface shape outside of this rectangular area. What knots must you add to U and V? State clearly how many and what values.

5.4. Consider the B-spline surface

$$\mathbf{S}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{2} N_{i,2}(u)N_{j,2}(v)\mathbf{P}_{i,j}$$

where $$U = \left\{0,0,0,\frac{1}{2},1,1,1\right\}$$

and $$V = \{0, 0, 0, 1, 1, 1\}$$

and
$$\mathbf{P}_{0,0} = (0, 0, 0) \quad \mathbf{P}_{1,0} = (3, 0, 3) \quad \mathbf{P}_{2,0} = (6, 0, 3) \quad \mathbf{P}_{3,0} = (9, 0, 0)$$
$$\mathbf{P}_{0,1} = (0, 2, 2) \quad \mathbf{P}_{1,1} = (3, 2, 5) \quad \mathbf{P}_{2,1} = (6, 2, 5) \quad \mathbf{P}_{3,1} = (9, 2, 2)$$
$$\mathbf{P}_{0,2} = (0, 4, 0) \quad \mathbf{P}_{1,2} = (3, 4, 3) \quad \mathbf{P}_{2,2} = (6, 4, 3) \quad \mathbf{P}_{3,2} = (9, 4, 0)$$

Compute $\mathbf{S}(3/10, 6/10)$ using knot insertion. Compare this with Exercise 3.8.

5.5. Let $p = 3$ and $U = \{0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2\}$ as in the example of Figure 5.26. Assume the control points $\mathbf{P}_i^0 = \{(-1, 0), (-1, 1), (-1/2, 3/2), (1/4, 3/2), (1, 3/2), (2, 1), (2, 0)\}$. Determine how many times $u = 1$ is removable, and compute the new control points. You can do this by using Eq. (5.29), or by tracing through Algorithm `A5.8`.