- developed a comprehensive NURBS library, called *Nlib V1.0, V2.0*. This library is the result of over 20 man-years of experience in NURBS research and development, and it combines new and well-tried software practices applied in previous systems that we designed;
- tested every single formula and algorithm, and presented graphical illustrations precisely computed using the routines of *Nlib*. **This book does not contain any hand-drawn figures; each figure is precisely computed and hence is accurate.**

We are pleased to present all of the accomplishments to the reader: (1) the book as a comprehensive reference, (2) *Nlib* source code (to order please see page 639 of this volume), and (3) the illustrations to instructors who adopt the book to teach a course on NURBS. In order for the reader to appreciate the enormous amount of work that went into this reference book, we present some data. To generate the graphical illustrations and to build *Nlib*, we wrote exactly (not counting the hundreds of test programs)

- **1,524** programs, that required
- **15,001,600** bytes of storage, which is roughly equivalent to
- **350,000** lines of code.

It was no picnic!

Some years ago a few researchers joked about NURBS, saying that the acronym really stands for **N**obody **U**nderstands **R**ational **B**-**S**plines. We admit that our colleagues were right. In the last four years, we were largely influenced by this interpretation and tried to present the material in the book in an intuitive manner. We hope that this helps change the acronym NURBS to EURBS, that is, **E**verybody **U**nderstands **R**ational **B**-**S**plines. We welcome the reader's opinion on our job and suggestions on possible improvements.

It is our pleasure to acknowledge the help and support of many people and organizations. First and foremost, we are grateful to our spouses, Karen Piegl and LaVella Tiller, for their patience, support, and love. We owe special thanks to Nancy Rogers of NAR Associates for the beautiful typesetting job, and David Rogers for the editorial and technical discussions that led to many improvements in the manuscript. We also thank Jim Oliver and Tim Strotman for the many suggestions and technical correspondence that helped shape this book into its current form. Tiller also thanks the many past and present colleagues in industry who over the years contributed inspiring discussions, valuable insights, support, and collegial companionship: They know who they are. Piegl's research was supported in part by the National Science Foundation under grant CCR-9217768 awarded to the University of South Florida, and by various grants from the Florida High Technology and Industry Council.

March 1995

Les Piegl
Wayne Tiller

CONTENTS

CHAPTER
ONE

Curve and Surface Basics

1.1 Implicit and Parametric Forms

The two most common methods of representing curves and surfaces in geometric modeling are implicit equations and parametric functions.

The implicit equation of a curve lying in the xy plane has the form $f(x, y) = 0$. This equation describes an implicit relationship between the x and y coordinates of the points lying on the curve. For a given curve the equation is unique up to a multiplicative constant. An example is the circle of unit radius centered at the origin, specified by the equation $f(x, y) = x^2 + y^2 - 1 = 0$ (Figure 1.1).

In parametric form, each of the coordinates of a point on the curve is represented separately as an explicit function of an independent parameter

$$\mathbf{C}(u) = \big(x(u), y(u)\big) \qquad a \le u \le b$$

Thus, $\mathbf{C}(u)$ is a vector-valued function of the independent variable, u. Although the interval $[a, b]$ is arbitrary, it is usually normalized to $[0, 1]$. The first quadrant of the circle shown in Figure 1.1 is defined by the parametric functions

$$\begin{aligned} x(u) &= \cos(u) \\ y(u) &= \sin(u) \qquad 0 \le u \le \frac{\pi}{2} \end{aligned} \tag{1.1}$$

Setting $t = \tan(u/2)$, one can derive the alternate representation

$$\begin{aligned} x(t) &= \frac{1 - t^2}{1 + t^2} \\ y(t) &= \frac{2t}{1 + t^2} \qquad 0 \le t \le 1 \end{aligned} \tag{1.2}$$

Thus, the parametric representation of a curve is not unique.

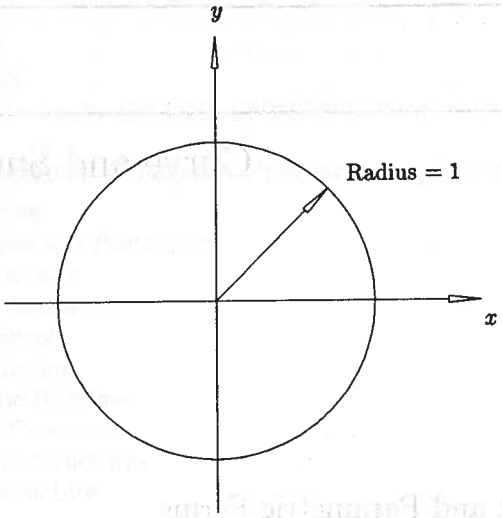


Figure 1.1. A circle of radius 1, centered at the origin.

It is instructive to think of $\mathbf{C}(u) = \big(x(u), y(u)\big)$ as the path traced out by a particle as a function of time; u is the time variable, and $[a, b]$ is the time interval. The first and second derivatives of $\mathbf{C}(u)$ are the velocity and acceleration of the particle, respectively. Differentiating Eqs. (1.1) and (1.2) once yields the velocity functions

$$\mathbf{C}'(u) = \big(x'(u), y'(u)\big) = \big(-\sin(u), \cos(u)\big)$$

$$\mathbf{C}'(t) = \big(x'(t), y'(t)\big) = \left(\frac{-4t}{(1+t^2)^2}, \frac{2(1-t^2)}{(1+t^2)^2}\right)$$

Notice that the magnitude of the velocity vector, $\mathbf{C}'(u)$, is a constant

$$|\mathbf{C}'(u)| = \sqrt{\sin^2(u) + \cos^2(u)} = 1$$

i.e., the direction of the particle is changing with time, but its speed is constant. This is referred to as a *uniform parameterization.* Substituting $t = 0$ and $t = 1$ into $\mathbf{C}'(t)$ yields $\mathbf{C}'(0) = (0, 2)$ and $\mathbf{C}'(1) = (-1, 0)$, i.e., the particle's starting speed is twice its ending speed (Figure 1.2).

A surface is defined by an implicit equation of the form $f(x, y, z) = 0$. An example is the sphere of unit radius centered at the origin, shown in Figure 1.3 and specified by the equation $x^2 + y^2 + z^2 - 1 = 0$. A parametric representation (not unique) of the same sphere is given by $\mathbf{S}(u, v) = \big(x(u, v), y(u, v), z(u, v)\big)$, where

$$\begin{aligned} x(u, v) &= \sin(u)\cos(v) \\ y(u, v) &= \sin(u)\sin(v) \\ z(u, v) &= \cos(u) \qquad 0 \le u \le \pi,\ \ 0 \le v \le 2\pi \end{aligned} \tag{1.3}$$

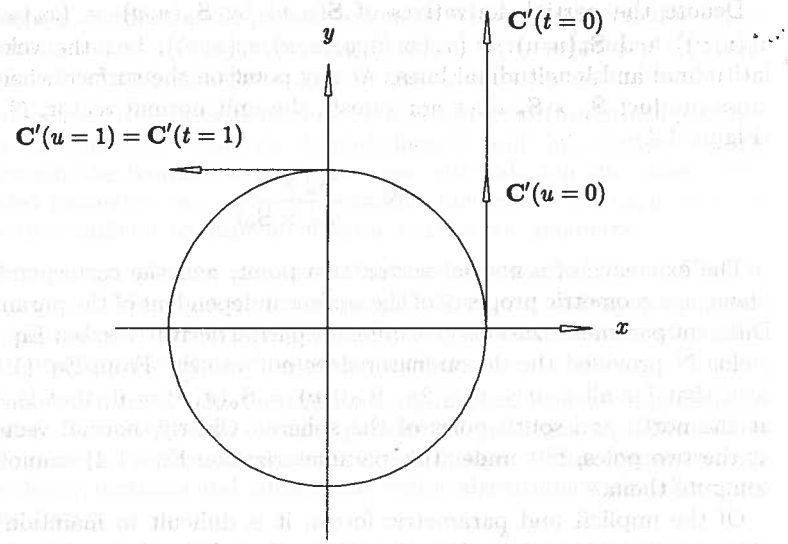


Figure 1.2. Velocity vectors $\mathbf{C}'(u)$ and $\mathbf{C}'(t)$ at $u, t = 0$, and 1.

Notice that two parameters are required to define a surface. Holding u fixed and varying v generates the latitudinal lines of the sphere; holding v fixed and varying u generates the longitudinal lines.

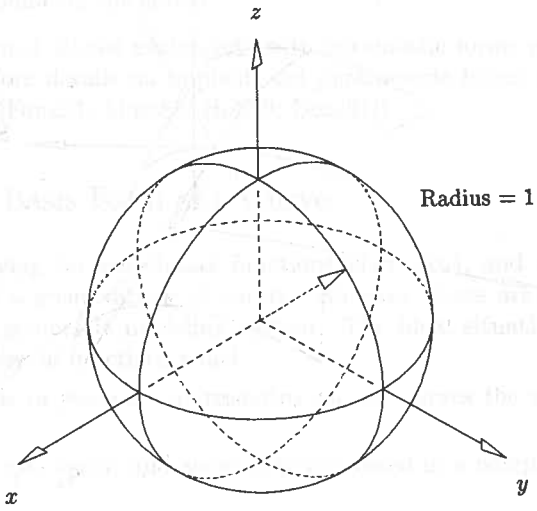


Figure 1.3. A sphere of radius 1, centered at the origin.

Denote the partial derivatives of $\mathbf{S}(u,v)$ by $\mathbf{S}_u(u,v) = \big(x_u(u,v), y_u(u,v), z_u(u,v)\big)$ and $\mathbf{S}_v(u,v) = \big(x_v(u,v), y_v(u,v), z_v(u,v)\big)$, i.e., the velocities along latitudinal and longitudinal lines. At any point on the surface where the vector cross product $\mathbf{S}_u \times \mathbf{S}_v$ does not vanish, the unit normal vector, $\mathbf{N}$, is given by (Figure 1.4)

$$\mathbf{N} = \frac{\mathbf{S}_u \times \mathbf{S}_v}{|\mathbf{S}_u \times \mathbf{S}_v|} \tag{1.4}$$

The existence of a normal vector at a point, and the corresponding tangent plane, is a geometric property of the surface independent of the parameterization. Different parameterizations give different partial derivatives, but Eq. (1.4) always yields $\mathbf{N}$ provided the denominator does not vanish. From Eq. (1.3) it can be seen that for all v, $0 \le v \le 2\pi$, $\mathbf{S}_v(0,v) = \mathbf{S}_v(\pi,v) = 0$, that is, $\mathbf{S}_v$ vanishes at the north and south poles of the sphere. Clearly, normal vectors do exist at the two poles, but under this parameterization Eq. (1.4) cannot be used to compute them.

Of the implicit and parametric forms, it is difficult to maintain that one is always more appropriate than the other. Both have their advantages and disadvantages. Successful geometric modeling is done using both techniques. A comparison of the two methods follows:

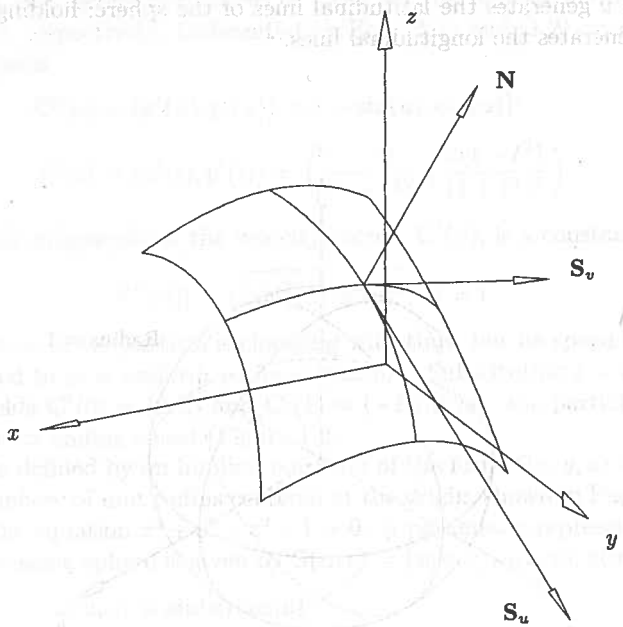


Figure 1.4. Partial derivative and unit normal vectors of $\mathbf{S}(u,v)$.

- By adding a z coordinate, the parametric method is easily extended to represent arbitrary curves in three-dimensional space, $\mathbf{C}(u) = \big(x(u), y(u), z(u)\big)$; the implicit form only specifies curves in the xy (or xz or yz) plane;
- It is cumbersome to represent bounded curve segments (or surface patches) with the implicit form. However, boundedness is built into the parametric form through the bounds on the parameter interval. On the other hand, unbounded geometry (e.g., a simple straight line given by $f(x,y) = ax + by + c = 0$) is difficult to implement using parametric geometry;
- Parametric curves possess a natural direction of traversal (from $\mathbf{C}(a)$ to $\mathbf{C}(b)$ if $a \le u \le b$); implicit curves do not. Hence, it is easy to generate ordered sequences of points along a parametric curve. A similar statement holds for generating meshes of points on surfaces;
- The parametric form is more natural for designing and representing shape in a computer. The coefficients of many parametric functions, e.g., Bézier and B-spline, possess considerable geometric significance. This translates into intuitive design methods and numerically stable algorithms with a distinctly geometric flavor;
- The complexity of many geometric operations and manipulations depends greatly on the method of representation. Two classic examples are:
  - compute a point on a curve or surface – difficult in the implicit form;
  - given a point, determine if it is on the curve or surface – difficult in the parametric form;
- In the parametric form, one must sometimes deal with parametric anomalies which are unrelated to true geometry. An example of this is the unit sphere (see Eq.[1.3]). The poles are parametric critical points which are algorithmically difficult, but geometrically the poles are no different than any other point on the sphere.

We are concerned almost exclusively with parametric forms in the remainder of this book. More details on implicit and parameteric forms can be found in standard texts ([Faux81; Mort85; Hoff89; Beac91]).

1.2 Power Basis Form of a Curve

Clearly, by allowing the coordinate functions $x(u)$, $y(u)$, and $z(u)$ to be arbitrary, we obtain a great variety of curves. However, there are trade-offs when implementing a geometric modeling system. The ideal situation is to restrict ourselves to a class of functions which

- are capable of precisely representing all the curves the users of the system need;
- are easily, efficiently, and accurately processed in a computer, in particular:
  - the computation of points and derivatives on the curves is efficient;

  - numerical processing of the functions is relatively insensitive to floating point round-off error;
  - the functions require little memory for storage;
- are simple and mathematically well understood.

A widely used class of functions is the polynomials. Although they satisfy the last two criteria in this list, there are a number of important curve (and surface) types which cannot be precisely represented using polynomials; these curves must be approximated in systems using polynomials. In this section and the next, we study two common methods of expressing polynomial functions, power basis and Bézier. Although mathematically equivalent, we will see that the Bézier method is far better suited to representing and manipulating shape in a computer.

An nth-degree power basis curve is given by

$$\mathbf{C}(u) = (x(u), y(u), z(u)) = \sum_{i=0}^{n} \mathbf{a}_i u^i \qquad 0 \le u \le 1 \tag{1.5}$$

The $\mathbf{a}_i = (x_i, y_i, z_i)$ are vectors, hence

$$x(u) = \sum_{i=0}^{n} x_i u^i \quad y(u) = \sum_{i=0}^{n} y_i u^i \quad z(u) = \sum_{i=0}^{n} z_i u^i$$

In matrix form Eq. (1.5) is

$$\mathbf{C}(u) = [\mathbf{a}_0 \quad \mathbf{a}_1 \quad \cdots \quad \mathbf{a}_n] \begin{bmatrix} 1 \\ u \\ \vdots \\ u^n \end{bmatrix} = [\mathbf{a}_i]^T [u^i] \tag{1.6}$$

(We write a row vector as the transpose of a column vector.)

Differentiating Eq. (1.5) yields

$$\mathbf{a}_i = \frac{\mathbf{C}^{(i)}(u)|_{u=0}}{i!}$$

where $\mathbf{C}^{(i)}(u)|_{u=0}$ is the ith derivative of $\mathbf{C}(u)$ at $u = 0$. The $n+1$ functions, $\{u^i\}$, are called the basis (or blending) functions, and the $\{\mathbf{a}_i\}$ the coefficients of the power basis representation.

Given u_0, the point $\mathbf{C}(u_0)$ on a power basis curve is most efficiently computed using Horner's method

- for degree $= 1$: $\mathbf{C}(u_0) = \mathbf{a}_1 u_0 + \mathbf{a}_0$
- degree $= 2$: $\mathbf{C}(u_0) = (\mathbf{a}_2 u_0 + \mathbf{a}_1) u_0 + \mathbf{a}_0$
- $\vdots$
- degree $= n$: $\mathbf{C}(u_0) = \left((\cdots(\mathbf{a}_n u_0 + \mathbf{a}_{n-1}) u_0 + \mathbf{a}_{n-2}\right) u_0 + \cdots + \mathbf{a}_0$

The general algorithm is

```
ALGORITHM A1.1
  Horner1(a,n,u0,C)
    { /*  Compute point on power basis curve.  */
      /*  Input:  a,n,u0  */
      /*  Output: C  */
    C = a[n];
    for (i=n-1; i>=0; i--)
      C = C*u0 + a[i];
    }
```

Examples

Ex1.1 $n = 1$. $\mathbf{C}(u) = \mathbf{a}_0 + \mathbf{a}_1 u$, $0 \le u \le 1$, is a straight line segment between the points $\mathbf{a}_0$ and $\mathbf{a}_0 + \mathbf{a}_1$ (Figure 1.5). The constant $\mathbf{C}'(u) = \mathbf{a}_1$ gives the direction of the line.

Ex1.2 $n = 2$. In general, $\mathbf{C}(u) = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2$, $0 \le u \le 1$, is a parabolic arc between the points $\mathbf{a}_0$ and $\mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_2$ (Figure 1.6). This is shown by

1. transforming $\mathbf{C}(u)$ into the xy plane ($\mathbf{C}(u)$ does lie in a unique plane);
2. setting $x = x_0 + x_1 u + x_2 u^2$ and $y = y_0 + y_1 u + y_2 u^2$, and then eliminating u and u^2 from these equations to obtain a second-degree implicit equation in x and y;
3. observing that the form of the implicit equation is that of a parabola.

Notice that the acceleration vector, $\mathbf{C}''(u) = 2\mathbf{a}_2$, is a constant. There are two special (degenerate) cases of interest, both occurring when the vector $\mathbf{a}_2$ is parallel to the initial tangent vector, $\mathbf{a}_1$ (when $x_1 y_2 = x_2 y_1$). In this case, the tangent vector does not turn, i.e., we get a straight line. The vector $\mathbf{a}_2$ can point in the same direction as $\mathbf{a}_1$ (Figure 1.7a), or in the opposite direction (Figure 1.7b). In Figure 1.7b, $\mathbf{a}_1 + 2\mathbf{a}_2 u_0 = 0$ for some $0 \le u_0 \le 1$ (velocity goes to zero, the particle stops), and a portion of the line segment is retraced in the opposite direction.

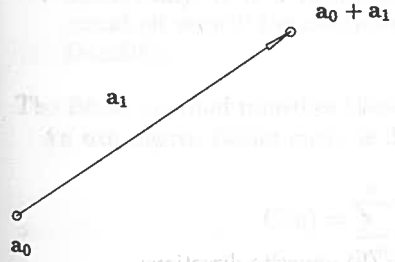


Figure 1.5. Straight line segment, $\mathbf{C}(u) = \mathbf{a}_0 + \mathbf{a}_1 u$.

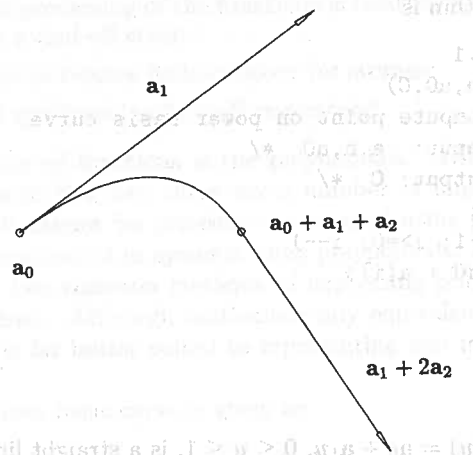
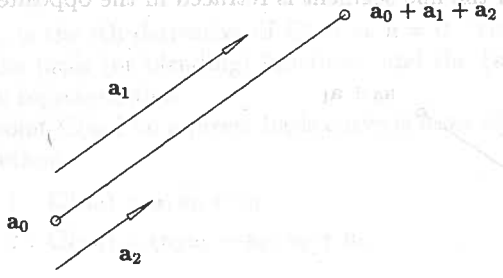


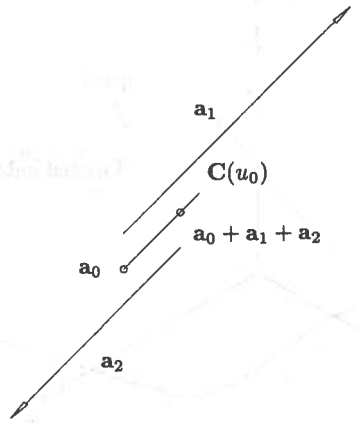
Figure 1.6. Parabolic arc, $\mathbf{C}(u) = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2$.

Ex1.3 $n = 3$. The cubic, $\mathbf{C}(u) = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2 + \mathbf{a}_3 u^3$, is a very general curve; it can be a truly *twisted* three-dimensional curve, not lying in a single plane (Figure 1.8a); it can have an inflection point (Figure 1.8b); a cusp (Figure 1.8c); or a loop (Figure 1.8d). A twisted curve results if $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ do not lie in a unique plane. An inflection point on a planar curve is defined as a point where the curve is smooth (no cusp) and the tangent line at that point passes through the curve. This implies a change in the turning direction of the curve. At an inflection point, either $\mathbf{C}''(u) = 0$, or $\mathbf{C}'(u) \parallel \mathbf{C}''(u)$. A necessary (but not sufficient) condition for a cusp at $u = u_0$ is $\mathbf{C}'(u_0) = 0$ (velocity zero). Conditions for a loop to occur are also known (see [Ferg66, 67, 69, 93; Forr70, 80; Wang81; Ston89; Su89]).



(a)

Figure 1.7. $\mathbf{a}_1$ and $\mathbf{a}_2$ parallel. (a) Same direction; (b) opposite directions.



(b)

Figure 1.7. (*Continued.*)

1.3 Bézier Curves

Next we study another parametric polynomial curve, the Bézier curve. Since they both use polynomials for their coordinate functions, the power basis and Bézier forms are mathematically equivalent; i.e., any curve that can be represented in one form can also be represented in the other form. However, the Bézier method is superior to the power basis form for geometric modeling. Our presentation of Bézier curves is rather informal; for a more rigorous and complete treatment the reader should consult other references [Forr72; Bezi72, 86; Gord74a; Chan81; Fari93; Yama88; Hosc93; Roge90].

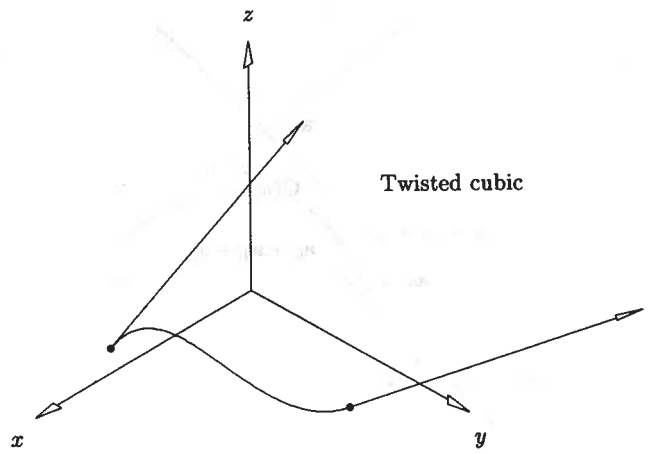
The power basis form has the following disadvantages:

- it is unnatural for interactive shape design; the coefficients $\{\mathbf{a}_i\}$ convey very little geometric insight about the shape of the curve. Furthermore, a designer typically wants to specify end conditions at both ends of the curve, not just at the starting point;
- algorithms for processing power basis polynomials have an algebraic rather than a geometric flavor (e.g., Horner's method);
- numerically, it is a rather poor form; e.g., Horner's method is prone to round-off error if the coefficients vary greatly in magnitude (see [Faro87, 88; Dani89]).

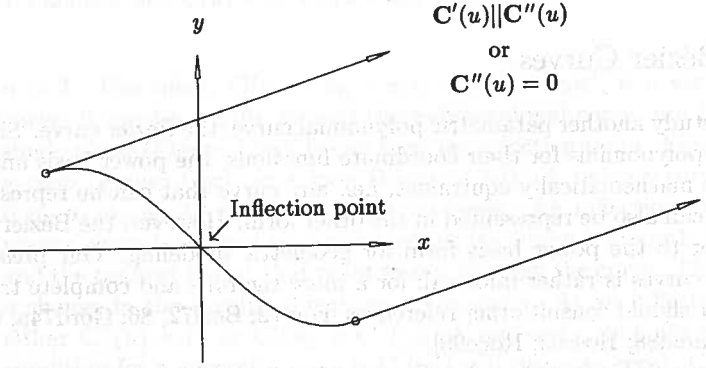
The Bézier method remedies these shortcomings.

An nth-degree Bézier curve is defined by

$$\mathbf{C}(u) = \sum_{i=0}^{n} B_{i,n}(u)\,\mathbf{P}_i \qquad 0 \le u \le 1 \tag{1.7}$$



(a)



(b)

Figure 1.8. Cubic curves. (a) Three-dimensional twisted; (b) inflection point; (c) cusp; (d) loop.

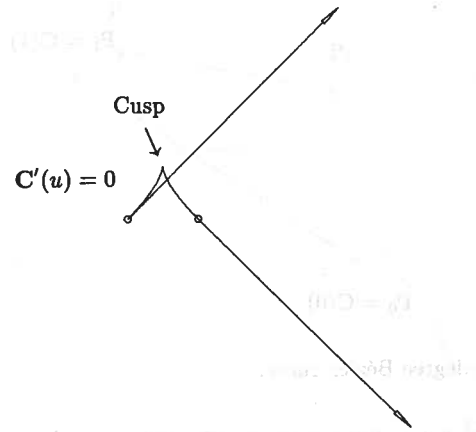
The basis (blending) functions, $\{B_{i,n}(u)\}$, are the classical nth-degree Bernstein polynomials ([Bern12; Lore86]) given by

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!}\, u^i\,(1-u)^{n-i} \tag{1.8}$$

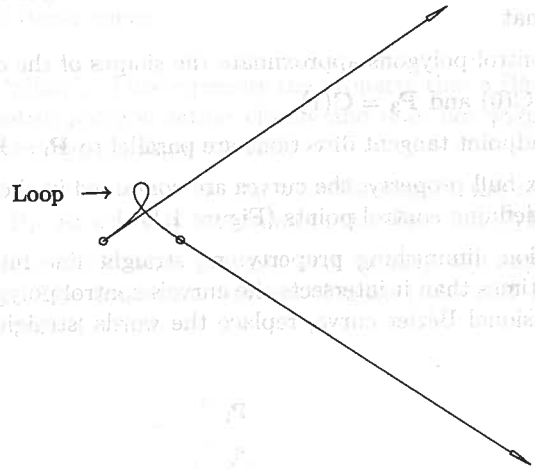
The geometric coefficients of this form, $\{\mathbf{P}_i\}$, are called *control points*. Notice that the definition, Eq. (1.7), requires that $u \in [0,1]$.

Examples

Ex1.4 $n = 1$. From Eq. (1.8) we have $B_{0,1}(u) = 1 - u$ and $B_{1,1}(u) = u$; and Eq. (1.7) takes the form $\mathbf{C}(u) = (1-u)\,\mathbf{P}_0 + u\mathbf{P}_1$. This is a straight line segment from $\mathbf{P}_0$ to $\mathbf{P}_1$ (see Figure 1.9).



(c)



(d)

Figure 1.8. (*Continued.*)

Ex1.5 $n = 2$. From Eqs. (1.7) and (1.8) we have $\mathbf{C}(u) = (1-u)^2\,\mathbf{P}_0 + 2u(1-u)\,\mathbf{P}_1 + u^2\,\mathbf{P}_2$. This is a parabolic arc from $\mathbf{P}_0$ to $\mathbf{P}_2$ (see Figure 1.10). Notice that

- the polygon formed by $\{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2\}$, called the *control polygon*, approximates the shape of the curve rather nicely;
- $\mathbf{P}_0 = \mathbf{C}(0)$ and $\mathbf{P}_2 = \mathbf{C}(1)$;
- the tangent directions to the curve at its endpoints are parallel to $\mathbf{P}_1 - \mathbf{P}_0$ and $\mathbf{P}_2 - \mathbf{P}_1$ (this is derived later);
- the curve is contained in the triangle formed by $\mathbf{P}_0\mathbf{P}_1\mathbf{P}_2$.

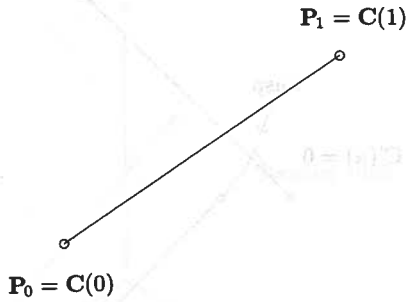


Figure 1.9. A first-degree Bézier curve.

Ex1.6 $n = 3$. We have $\mathbf{C}(u) = (1-u)^3\,\mathbf{P}_0 + 3u(1-u)^2\,\mathbf{P}_1 + 3u^2(1-u)\,\mathbf{P}_2 + u^3\,\mathbf{P}_3$. Examples of cubic Bézier curves are shown in Figures 1.11a to 1.11f. Notice that

- the control polygons approximate the shapes of the curves;
- $\mathbf{P}_0 = \mathbf{C}(0)$ and $\mathbf{P}_3 = \mathbf{C}(1)$;
- the endpoint tangent directions are parallel to $\mathbf{P}_1 - \mathbf{P}_0$ and $\mathbf{P}_3 - \mathbf{P}_2$;
- convex hull property: the curves are contained in the convex hulls of their defining control points (Figure 1.11c);
- variation diminishing property: no straight line intersects a curve more times than it intersects the curve's control polygon (for a three-dimensional Bézier curve, replace the words 'straight line' with the word 'plane'). This expresses the property that a Bézier curve follows its control polygon rather closely and does not wiggle more than its control polygon (Figure 1.11f);
- initially (at $u = 0$) the curve is turning in the same direction as $\mathbf{P}_0\mathbf{P}_1\mathbf{P}_2$. At $u = 1$ it is turning in the direction $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$;
- a loop in the control polygon may or may not imply a loop in the curve. The transition between Figure 1.11e and Figure 1.11f is a curve with a cusp.

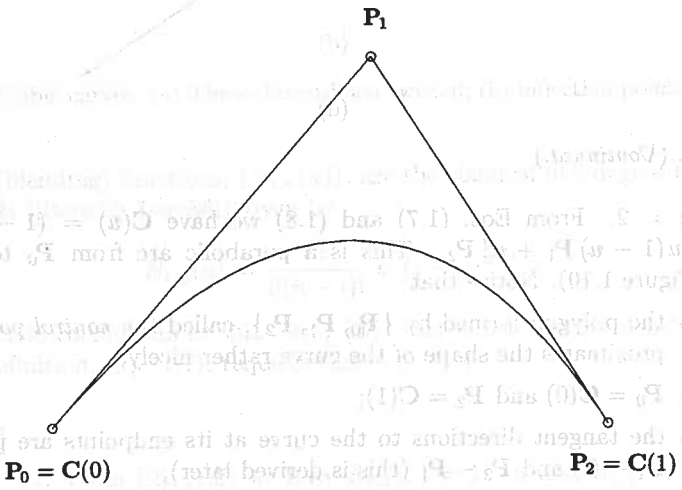


Figure 1.10. A second-degree Bézier curve.

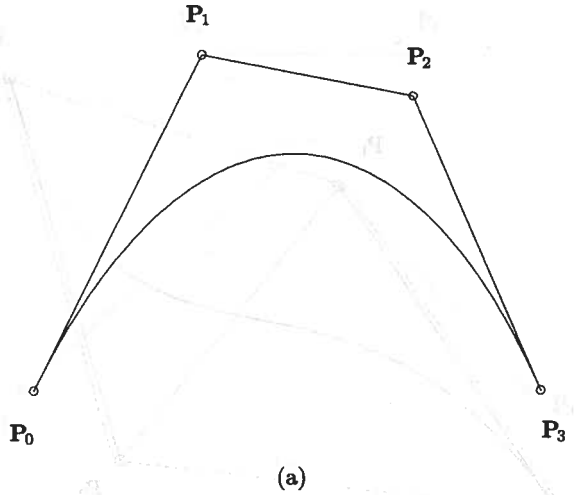


Figure 1.11. Cubic Bézier curves.

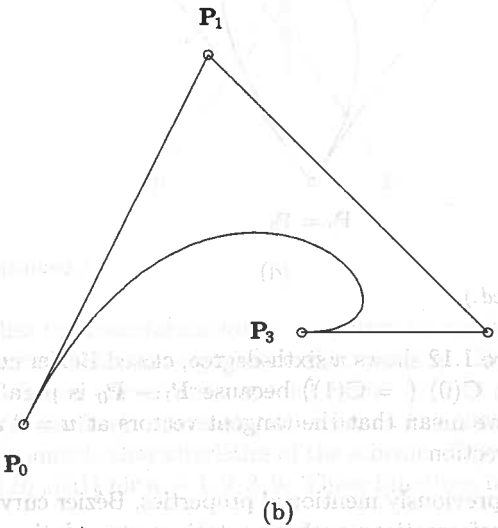
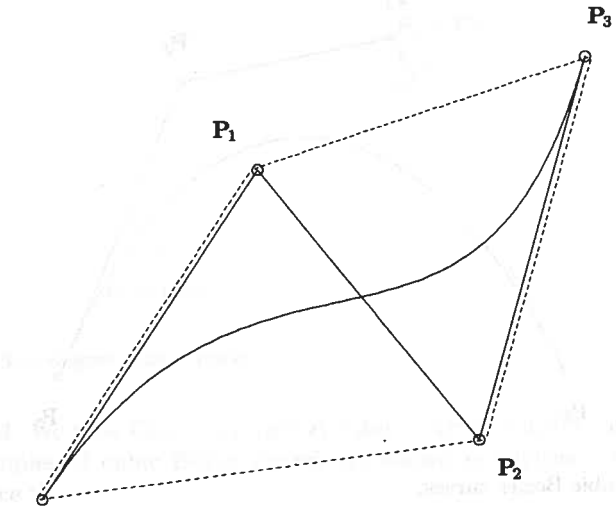
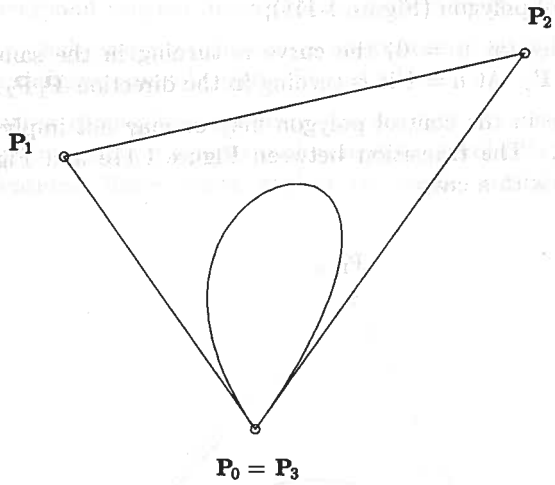


Figure 1.11. (*Continued.*)



(c)

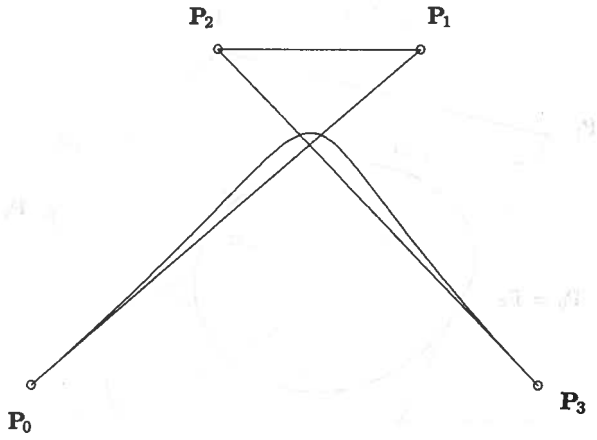


(d)

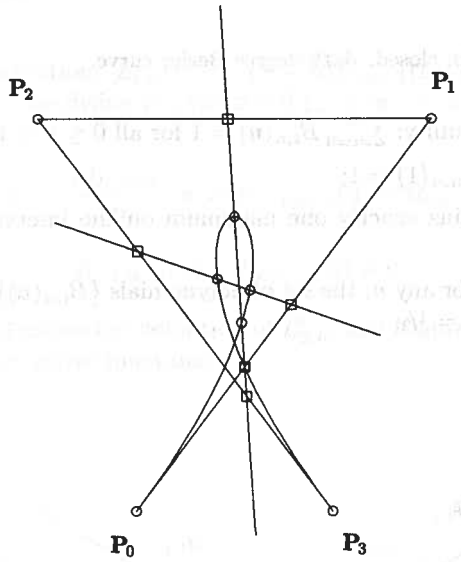
Figure 1.11. (*Continued.*)

Ex1.7 $n = 6$. Figure 1.12 shows a sixth-degree, closed Bézier curve. The curve is smooth at $\mathbf{C}(0)$ $(= \mathbf{C}(1))$ because $\mathbf{P}_1 - \mathbf{P}_0$ is parallel to $\mathbf{P}_6 - \mathbf{P}_5$. By smooth we mean that the tangent vectors at $u = 0$ and $u = 1$ have the same direction.

In addition to the previously mentioned properties, Bézier curves are invariant under the usual transformations such as rotations, translations, and scalings;



(e)



(f)

Figure 1.11. (*Continued.*)

that is, one applies the transformation to the curve by applying it to the control polygon. We present this concept more rigorously in Chapter 3 for B-spline curves (of which Bézier curves are a special case).

In any curve (or surface) representation scheme, the choice of basis functions determines the geometric characteristics of the scheme. Figures 1.13a–d show the basis functions $\{B_{i,n}(u)\}$ for $n = 1, 2, 3, 9$. These functions have these properties:

P1.1 nonnegativity: $B_{i,n}(u) \geq 0$ for all i, n and $0 \leq u \leq 1$;

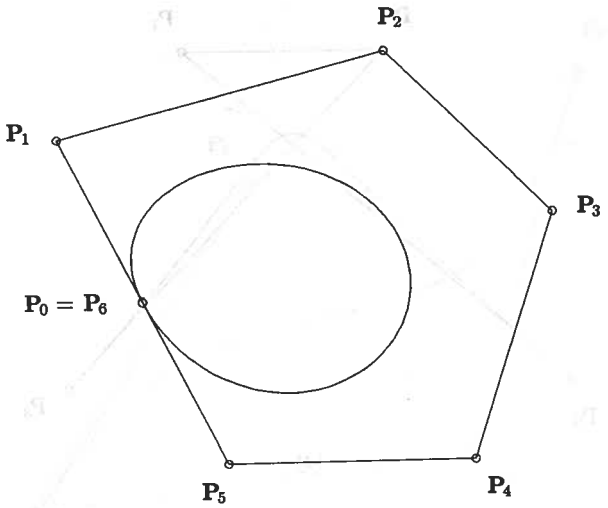


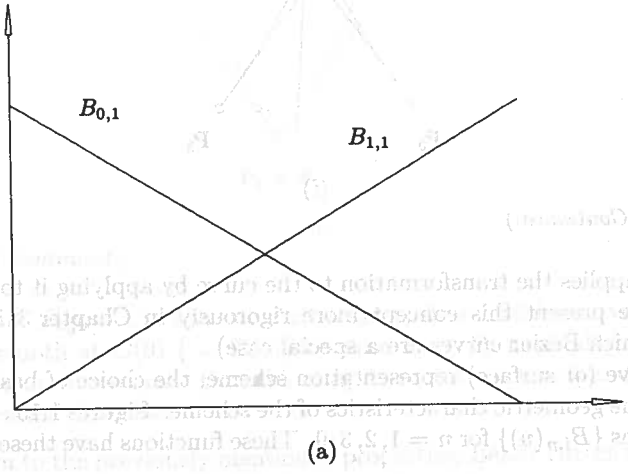
Figure 1.12. A smooth, closed, sixth-degree Bézier curve.

P1.2 partition of unity: $\sum_{i=0}^{n} B_{i,n}(u) = 1$ for all $0 \le u \le 1$;

P1.3 $B_{0,n}(0) = B_{n,n}(1) = 1$;

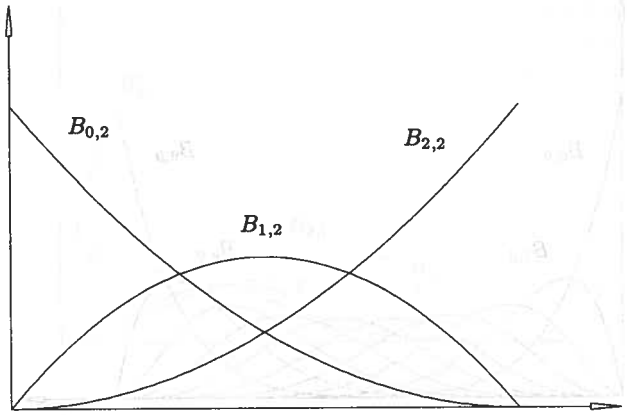
P1.4 $B_{i,n}(u)$ attains exactly one maximum on the interval $[0,1]$, that is, at $u = i/n$;

P1.5 symmetry: for any n, the set of polynomials $\{B_{i,n}(u)\}$ is symmetric with respect to $u = 1/2$;



(a)

Figure 1.13. The Bernstein polynomials for (a) $n = 1$; (b) n = 2; (c) $n = 3$; (d) n = 9.



(b)

Figure 1.13. (*Continued.*)

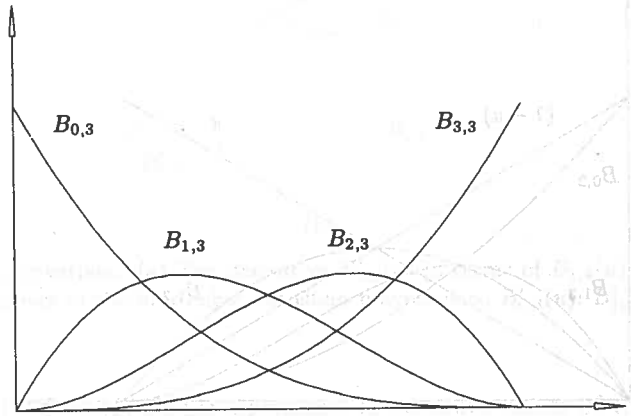
P1.6 recursive definition: $B_{i,n}(u) = (1-u)B_{i,n-1}(u) + uB_{i-1,n-1}(u)$ (see Figure 1.14); we define $B_{i,n}(u) \equiv 0$ if $i < 0$ or $i > n$;

P1.7 derivatives:

$$B'_{i,n}(u) = \frac{dB_{i,n}(u)}{du} = n\big(B_{i-1,n-1}(u) - B_{i,n-1}(u)\big)$$

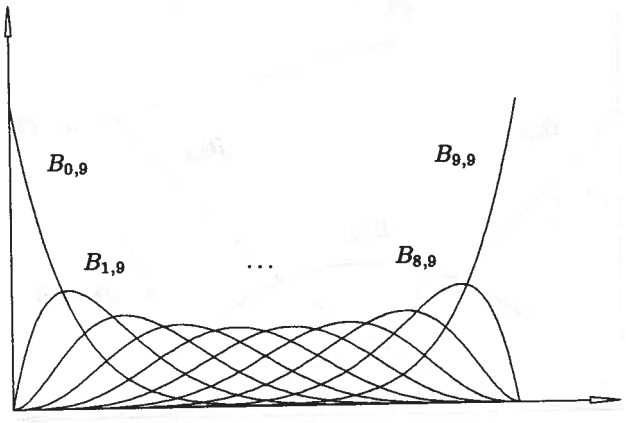
with $$B_{-1,n-1}(u) \equiv B_{n,n-1}(u) \equiv 0$$

Figure 1.15a shows the definition of $B'_{2,5}$, and Figure 1.15b illustrates all the cubic derivative functions.



(c)

Figure 1.13. (*Continued.*)



(d)

Figure 1.13. (*Continued.*)

From Eq. (1.8) we have $B_{0,0}(u) = 1$. Using property `P1.6`, the linear and quadratic Bernstein polynomials are

$$B_{0,1}(u) = (1-u)B_{0,0}(u) + uB_{-1,0}(u) = 1-u$$
$$B_{1,1}(u) = (1-u)B_{1,0}(u) + uB_{0,0}(u) = u$$
$$B_{0,2}(u) = (1-u)B_{0,1}(u) + uB_{-1,1}(u) = (1-u)^2$$
$$B_{1,2}(u) = (1-u)B_{1,1}(u) + uB_{0,1}(u) = (1-u)u + u(1-u) = 2u(1-u)$$
$$B_{2,2}(u) = (1-u)B_{2,1}(u) + uB_{1,1}(u) = u^2$$

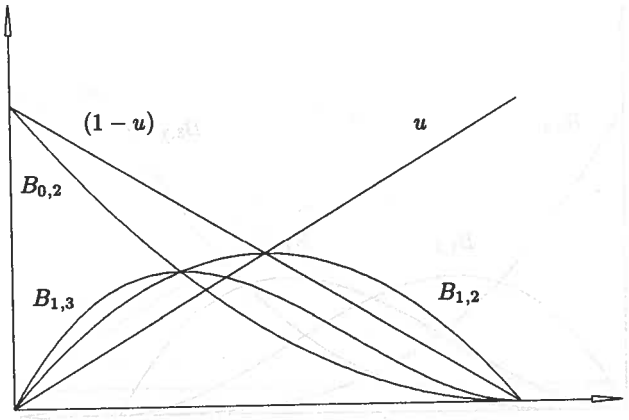
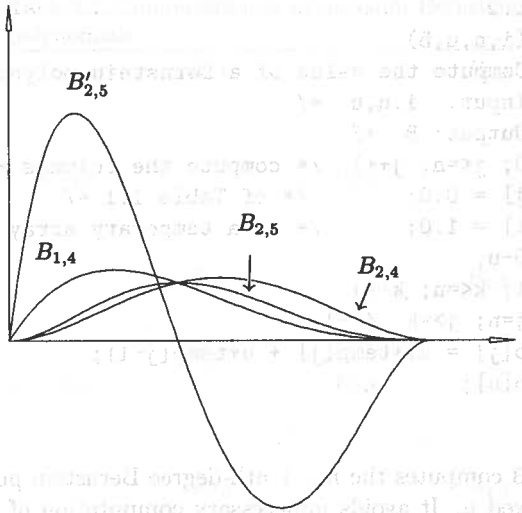
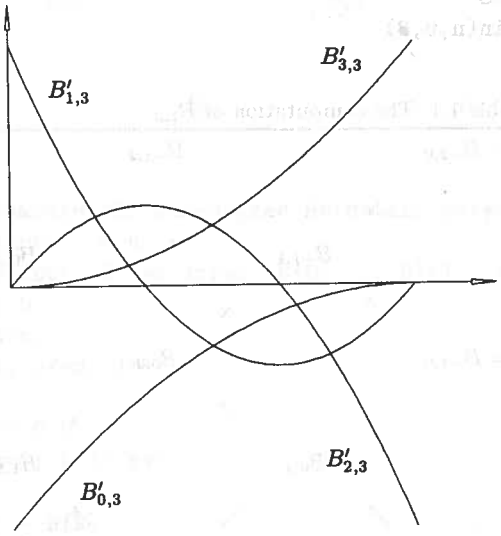


Figure 1.14. The recursive definition of the Bernstein polynomial, $B_{1,3}(u)$.



(a)



(b)

Figure 1.15. Derivatives. (a) The derivative $B'_{2,5}(u)$ in terms of $B_{1,4}(u)$ and $B_{2,4}(u)$; (b) the derivatives of the four cubic Bernstein polynomials, $B'_{0,3}(u)$; $B'_{1,3}(u)$; $B'_{2,3}(u)$; $B'_{3,3}(u)$.

Property `P1.6` yields simple algorithms to compute values of the Bernstein polynomials at fixed values of u. Algorithm `A1.2` computes the value $B_{i,n}(u)$ for fixed u. The computation of $B_{1,3}$ is depicted in Table 1.1.

```
ALGORITHM A1.2
  Bernstein(i,n,u,B)
    {  /*  Compute the value of a Bernstein polynomial.  */
       /*  Input:  i,n,u  */
       /*  Output: B  */
      for (j=0; j<=n; j++)  /* compute the columns */
        temp[j] = 0.0;       /* of Table 1.1 */
      temp[n-i] = 1.0;      /* in a temporary array */
      u1 = 1.0-u;
      for (k=1; k<=n; k++)
        for (j=n; j>=k; j--)
          temp[j] = u1*temp[j] + u*temp[j-1];
      B = temp[n];
      }
```

Algorithm A1.3 computes the $n+1$ nth-degree Bernstein polynomials which are nonzero at fixed u. It avoids unnecessary computation of zero terms. The algorithm is depicted in Table 1.2 for the cubic case.

```
ALGORITHM A1.3
  AllBernstein(n,u,B)
```

Table 1.1. The computation of $B_{1,3}$.

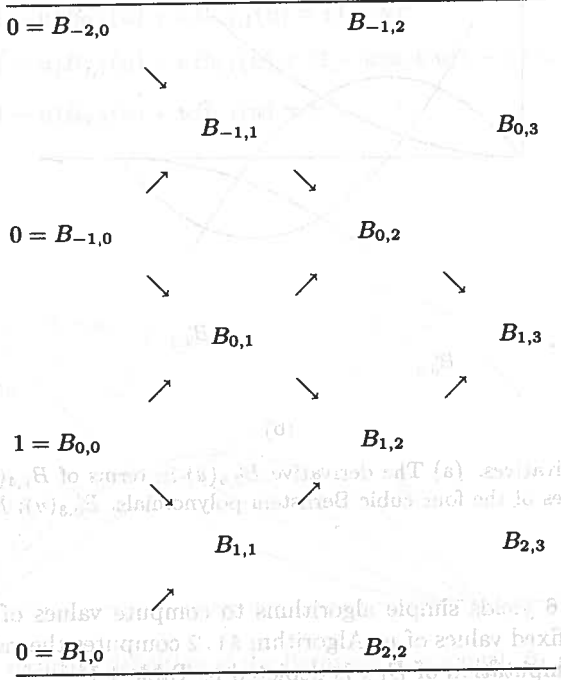
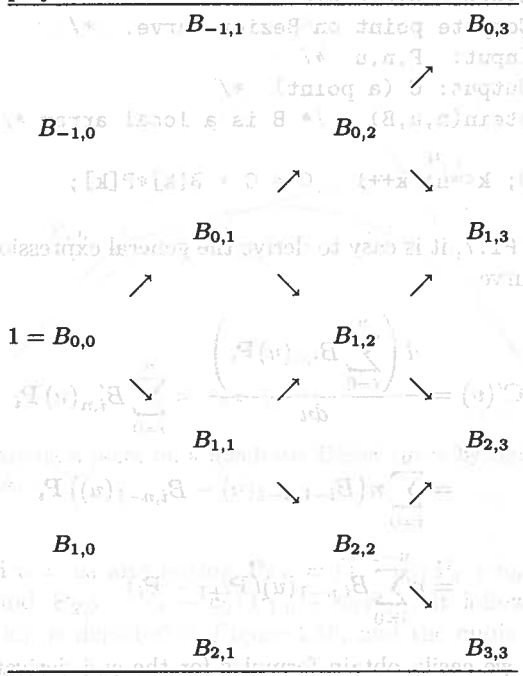


Table 1.2. Computation of all the cubic Bernstein polynomials.




```
{  /*  Compute all nth-degree Bernstein polynomials.  */
   /*  Input:  n,u  */
   /*  Output: B (an array, B[0],...,B[n])  */
B[0] = 1.0;
u1 = 1.0-u;
for (j=1; j<=n; j++)
  {
  saved = 0.0;
  for (k=0; k<j; k++)
    {
    temp = B[k];
    B[k] = saved+u1*temp;
    saved = u*temp;
    }
  B[j] = saved;
  }
}
```

Algorithm A1.4 combines A1.3 and Eq. (1.7) to compute the point on an nth-degree Bézier curve at a fixed u value.

```
ALGORITHM A1.4
  PointOnBezierCurve(P,n,u,C)
    { /*  Compute point on Bezier curve.  */
      /*  Input:  P,n,u  */
      /*  Output: C (a point)  */
    AllBernstein(n,u,B)   /* B is a local array */
    C = 0.0;
    for (k=0; k<=n; k++)    C = C + B[k]*P[k];
    }
```

Using property P1.7, it is easy to derive the general expression for the derivative of a Bézier curve

$$\begin{aligned}\mathbf{C}'(u) &= \frac{d\left(\sum_{i=0}^{n} B_{i,n}(u)\,\mathbf{P}_i\right)}{du} = \sum_{i=0}^{n} B'_{i,n}(u)\,\mathbf{P}_i \\ &= \sum_{i=0}^{n} n\big(B_{i-1,n-1}(u) - B_{i,n-1}(u)\big)\,\mathbf{P}_i \\ &= n\sum_{i=0}^{n-1} B_{i,n-1}(u)(\mathbf{P}_{i+1} - \mathbf{P}_i) \qquad (1.9)\end{aligned}$$

From Eq. (1.9) we easily obtain formulas for the end derivatives of a Bézier curve, e.g.

$$\begin{aligned}\mathbf{C}'(0) &= n(\mathbf{P}_1 - \mathbf{P}_0) \qquad & \mathbf{C}''(0) &= n(n-1)(\mathbf{P}_0 - 2\mathbf{P}_1 + \mathbf{P}_2) \\ \mathbf{C}'(1) &= n(\mathbf{P}_n - \mathbf{P}_{n-1}) \qquad & \mathbf{C}''(1) &= n(n-1)(\mathbf{P}_n - 2\mathbf{P}_{n-1} + \mathbf{P}_{n-2}) \qquad (1.10)\end{aligned}$$

Notice from Eqs. (1.9) and (1.10) that

- the derivative of an nth-degree Bézier curve is an $(n-1)$th-degree Bézier curve;
- the expressions for the end derivatives at $u = 0$ and $u = 1$ are symmetric (due, of course, to the symmetry of the basis functions);
- the kth derivative at an endpoint depends (in a geometrically very intuitive manner) solely on the $k+1$ control points at that end.

Let $n = 2$ and $\mathbf{C}(u) = \sum_{i=0}^{2} B_{i,2}(u)\,\mathbf{P}_i$. Then

$$\begin{aligned}\mathbf{C}(u) &= (1-u)^2\,\mathbf{P}_0 + 2u(1-u)\,\mathbf{P}_1 + u^2\,\mathbf{P}_2 \\ &= (1-u)\big(\underbrace{(1-u)\,\mathbf{P}_0 + u\,\mathbf{P}_1}_{\text{linear}}\big) + u\big(\underbrace{(1-u)\,\mathbf{P}_1 + u\,\mathbf{P}_2}_{\text{linear}}\big)\end{aligned}$$

Thus, $\mathbf{C}(u)$ is obtained as the linear interpolation of two first-degree Bézier curves; in particular, any point on $\mathbf{C}(u)$ is obtained by three linear interpolations.

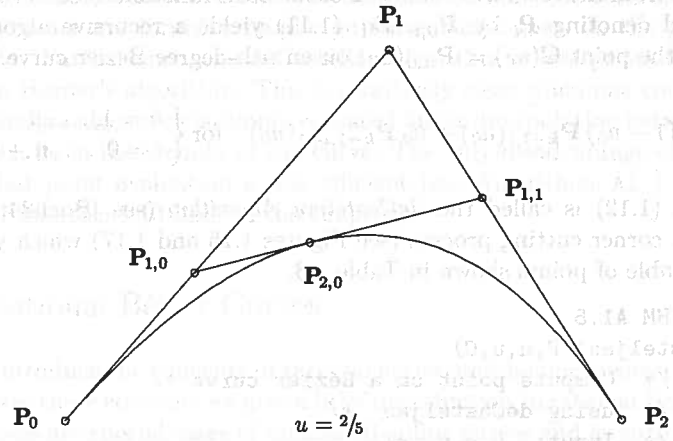


Figure 1.16. Obtaining a point on a quadratic Bézier curve by repeated linear interpolation at $u_0 = 2/5$.

Assuming a fixed $u = u_0$ and letting $\mathbf{P}_{1,0} = (1-u_0)\,\mathbf{P}_0 + u_0\,\mathbf{P}_1$, $\mathbf{P}_{1,1} = (1-u_0)\,\mathbf{P}_1 + u_0\,\mathbf{P}_2$, and $\mathbf{P}_{2,0} = (1-u_0)\,\mathbf{P}_{1,0} + u_0\,\mathbf{P}_{1,1}$, it follows that $\mathbf{C}(u_0) = \mathbf{P}_{2,0}$. The situation is depicted in Figure 1.16, and the cubic case is shown in Figure 1.17.

Denoting a general nth-degree Bézier curve by $\mathbf{C}_n(\mathbf{P}_0, \ldots, \mathbf{P}_n)$, we have

$$\begin{aligned}\mathbf{C}_n(\mathbf{P}_0, \ldots, \mathbf{P}_n) &= (1-u)\mathbf{C}_{n-1}(\mathbf{P}_0, \ldots, \mathbf{P}_{n-1}) \\ &\quad + u\mathbf{C}_{n-1}(\mathbf{P}_1, \ldots, \mathbf{P}_n) \qquad (1.11)\end{aligned}$$

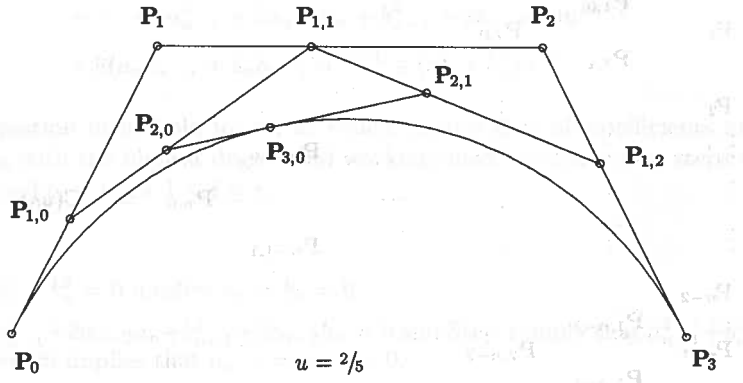


Figure 1.17. A point on a cubic Bézier curve by repeated linear interpolation at $u_0 = 2/5$.

This follows from the recursive definition of the basis functions (see P1.6). Fixing $u = u_0$ and denoting $\mathbf{P}_i$ by $\mathbf{P}_{0,i}$, Eq. (1.11) yields a recursive algorithm for computing the point $\mathbf{C}(u_0) = \mathbf{P}_{n,0}(u_0)$ on an nth-degree Bézier curve, i.e.

$$\mathbf{P}_{k,i}(u_0) = (1-u_0)\,\mathbf{P}_{k-1,i}(u_0) + u_0\,\mathbf{P}_{k-1,i+1}(u_0) \quad \text{for} \begin{cases} k = 1,\ldots,n \\ i = 0,\ldots,n-k \end{cases} \tag{1.12}$$

Equation (1.12) is called the *deCasteljau Algorithm* (see [Boeh84; deCa86, 93]). It is a corner cutting process (see Figures 1.16 and 1.17) which yields the triangular table of points shown in Table 1.3.

```
ALGORITHM A1.5
  deCasteljau1(P,n,u,C)
    { /*  Compute point on a Bézier curve */
      /*  using deCasteljau  */
      /*  Input:  P,n,u  */
      /*  Output: C (a point)  */
    for (i=0; i<=n; i++)    /* Use local array so we do not */
      Q[i] = P[i];          /* destroy control points */
    for (k=1; k<=n; k++)
      for (i=0; i<=n-k; i++)
        Q[i] = (1.0-u)*Q[i] + u*Q[i+1];
    C = Q[0];
    }
```

We conclude this section with a comparison of the Bézier and power basis methods. Clearly, the Bézier form is the more geometric of the two. Equation (1.10), together with the convex hull and variation diminishing properties, makes

Table 1.3. Points generated by the deCasteljau algorithm.

$\mathbf{P}_0$					
	$\mathbf{P}_{1,0}$				
$\mathbf{P}_1$		$\mathbf{P}_{2,0}$			
	$\mathbf{P}_{1,1}$				
$\mathbf{P}_2$		⋮			
⋮	⋮	⋮	$\mathbf{P}_{n-1,0}$		
⋮	⋮	⋮	⋯	$\mathbf{P}_{n,0}$ =	$\mathbf{C}(u_0)$
⋮	⋮	⋮	$\mathbf{P}_{n-1,1}$		
$\mathbf{P}_{n-2}$		⋮			
	$\mathbf{P}_{1,n-2}$				
$\mathbf{P}_{n-1}$		$\mathbf{P}_{2,n-2}$			
	$\mathbf{P}_{1,n-1}$				
$\mathbf{P}_n$					

Bézier curves more suitable for interactive curve design. The control points give the designer a more intuitive handle on curve shape than do the power basis coefficients. Furthermore, the deCasteljau algorithm is less prone to round-off error than Horner's algorithm. This is intuitively clear when one considers that the deCasteljau algorithm is simply repeated linear interpolation between points, all of which lie in the vicinity of the curve. The only disadvantage of the Bézier form is that point evaluation is less efficient (see Algorithms A1.1, A1.4, and A1.5, and Exercise 1.13 later in the chapter).

1.4 Rational Bézier Curves

Next we introduce the concepts of rational curves and homogeneous coordinates. To illustrate these concepts we give a brief introduction to rational Bézier curves. These curves are special cases of rational B-spline curves and as such are treated more completely and rigorously in subsequent chapters.

Although polynomials offer many advantages, there exist a number of important curve and surface types which cannot be represented precisely using polynomials, e.g., circles, ellipses, hyperbolas, cylinders, cones, spheres, etc. As an example, we give a proof that the unit circle in the xy plane, centered at the origin, cannot be represented using polynomial coordinate functions. To the contrary, let us assume that

$$x(u) = a_0 + a_1u + \cdots + a_nu^n$$
$$y(u) = b_0 + b_1u + \cdots + b_nu^n$$

Then $x^2 + y^2 - 1 = 0$ implies that

$$\begin{aligned} 0 &= (a_0 + a_1u + \cdots + a_nu^n)^2 + (b_0 + b_1u + \cdots + b_nu^n)^2 - 1 \\ &= (a_0^2 + b_0^2 - 1) + 2(a_0a_1 + b_0b_1)u + (a_1^2 + 2a_0a_2 + b_1^2 + 2b_0b_2)u^2 \\ &\quad + \cdots + (a_{n-1}^2 + 2a_{n-2}a_n + b_{n-1}^2 + 2b_{n-2}b_n)u^{2n-2} \\ &\quad + 2(a_na_{n-1} + b_nb_{n-1})u^{2n-1} + (a_n^2 + b_n^2)u^{2n} \end{aligned}$$

This equation must hold for all u, which implies that all coefficients are zero. Starting with the highest degree and working down, we show in n steps that all $a_i = 0$ and $b_i = 0$ for $1 \le i \le n$.

Step

1. $a_n^2 + b_n^2 = 0$ implies $a_n = b_n = 0$.
2. $a_{n-1}^2 + 2a_{n-2}a_n + b_{n-1}^2 + 2b_{n-2}b_n = 0$ and Step 1 imply that $a_{n-1}^2 + b_{n-1}^2 = 0$ which implies that $a_{n-1} = b_{n-1} = 0$.

⋮

n. $a_1^2 + 2a_0a_2 + b_1^2 + 2b_0b_2 = 0$ and Step $n-1$ imply that $a_1^2 + b_1^2 = 0$, which implies that $a_1 = b_1 = 0$.

Thus, $x(u) = a_0$ and $y(u) = b_0$, which is an obvious contradiction.

It is known from classical mathematics that all the conic curves, including the circle, can be represented using *rational functions*, which are defined as the ratio of two polynomials. In fact, they are represented with rational functions of the form

$$x(u) = \frac{X(u)}{W(u)} \qquad y(u) = \frac{Y(u)}{W(u)} \tag{1.13}$$

where $X(u)$, $Y(u)$, and $W(u)$ are polynomials, that is, each of the coordinate functions has the same denominator.

Examples

Ex1.8 Circle of radius 1, centered at the origin

$$x(u) = \frac{1-u^2}{1+u^2} \qquad y(u) = \frac{2u}{1+u^2}$$

Ex1.9 Ellipse, centered at the origin; the y-axis is the major axis, the x-axis is the minor axis, and the major and minor radii are 2 and 1, respectively

$$x(u) = \frac{1-u^2}{1+u^2} \qquad y(u) = \frac{4u}{1+u^2}$$

Ex1.10 Hyperbola, center at $\mathbf{P} = (0, 4/3)$; the y-axis is the transverse axis

$$x(u) = \frac{-1+2u}{1+2u-2u^2} \qquad y(u) = \frac{4u(1-u)}{1+2u-2u^2}$$

The lower branch (with vertex at $\mathbf{P} = (0, 2/3)$) is traced out for

$$u \in \left(\frac{1-\sqrt{3}}{2}, \frac{1+\sqrt{3}}{2}\right)$$

Ex1.11 Parabola, vertex at the origin; the y-axis is the axis of symmetry

$$x(u) = u \qquad y(u) = u^2$$

Notice that the parabola does not require rational functions. The reader should sketch these functions. For the circle equations it is easy to see that for any u, $(x(u), y(u))$ lies on the unit circle centered at the origin

$$\big(x(u)\big)^2 + \big(y(u)\big)^2 = \left(\frac{1-u^2}{1+u^2}\right)^2 + \left(\frac{2u}{1+u^2}\right)^2$$

$$= \frac{1-2u^2+u^4+4u^2}{(1+u^2)^2} = \frac{(1+u^2)^2}{(1+u^2)^2} = 1$$

Define an nth-degree *rational Bézier curve* by (see [Forr68; Pieg86; Fari83, 89])

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i \mathbf{P}_i}{\sum_{i=0}^{n} B_{i,n}(u) w_i} \qquad 0 \le u \le 1 \tag{1.14}$$

The $\mathbf{P}_i = (x_i, y_i, z_i)$ and $B_{i,n}(u)$ are as before; the w_i are scalars, called the *weights*. Thus, $W(u) = \sum_{i=0}^{n} B_{i,n}(u) w_i$ is the common denominator function. Except where explicitly stated otherwise, we assume that $w_i > 0$ for all i. This ensures that $W(u) > 0$ for all $u \in [0,1]$. We write

$$\mathbf{C}(u) = \sum_{i=0}^{n} R_{i,n}(u) \mathbf{P}_i \qquad 0 \le u \le 1 \tag{1.15}$$

where

$$R_{i,n}(u) = \frac{B_{i,n}(u) w_i}{\sum_{j=0}^{n} B_{j,n}(u) w_j}$$

The $R_{i,n}(u)$ are the rational basis functions for this curve form. Figure 1.18a shows an example of cubic basis functions, and Figure 1.18b a corresponding cubic rational Bézier curve.

The $R_{i,n}(u)$ have properties which can be easily derived from Eq. (1.15) and the corresponding properties of the $B_{i,n}(u)$:

P1.8 nonnegativity: $R_{i,n}(u) \ge 0$ for all i, n and $0 \le u \le 1$;

P1.9 partition of unity: $\sum_{i=0}^{n} R_{i,n}(u) = 1$ for all $0 \le u \le 1$;

P1.10 $R_{0,n}(0) = R_{n,n}(1) = 1$;

P1.11 $R_{i,n}(u)$ attains exactly one maximum on the interval $[0,1]$;

P1.12 if $w_i = 1$ for all i, then $R_{i,n}(u) = B_{i,n}(u)$ for all i; i.e., the $B_{i,n}(u)$ are a special case of the $R_{i,n}(u)$.

These yield the following geometric properties of rational Bézier curves:

P1.13 convex hull property: the curves are contained in the convex hulls of their defining control points (the $\mathbf{P}_i$);

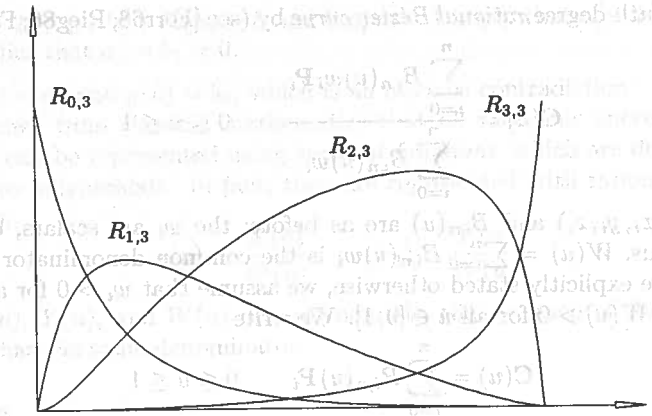
P1.14 transformation invariance: rotations, translations, and scalings are applied to the curve by applying them to the control points;

P1.15 variation diminishing property: same as for polynomial Bézier curves (see previous section);

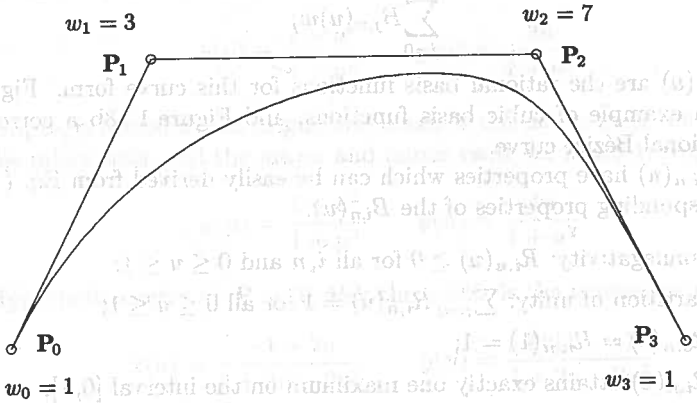
P1.16 endpoint interpolation: $\mathbf{C}(0) = \mathbf{P}_0$ and $\mathbf{C}(1) = \mathbf{P}_n$;

P1.17 the kth derivative at $u = 0$ ($u = 1$) depends on the first (last) $k+1$ control points and weights; in particular, $\mathbf{C}'(0)$ and $\mathbf{C}'(1)$ are parallel to $\mathbf{P}_1 - \mathbf{P}_0$ and $\mathbf{P}_n - \mathbf{P}_{n-1}$, respectively;

P1.18 polynomial Bézier curves are a special case of rational Bézier curves.



(a)



(b)

Figure 1.18. Rational cubic. (a) Basis functions; (b) Bézier curve.

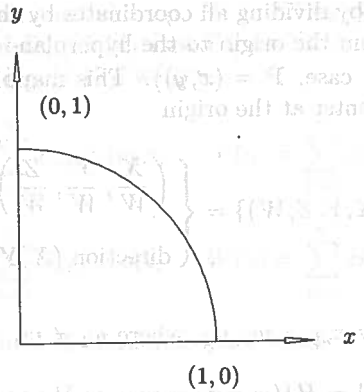
Example

Ex1.12 Let us consider the rational Bézier circular arc.

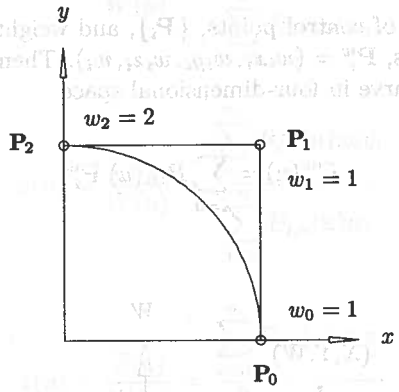
$$\mathbf{C}(u) = \big(x(u), y(u)\big) = \left(\frac{1-u^2}{1+u^2}, \frac{2u}{1+u^2}\right) \qquad 0 \le u \le 1$$

represents one quadrant of the unit circle, as shown in Figure 1.19a. We now derive the quadratic rational Bézier representation of this circular arc. Clearly, from P1.16 and P1.17, $\mathbf{P}_0 = (1,0)$, $\mathbf{P}_1 = (1,1)$, and $\mathbf{P}_2 = (0,1)$. For the weights we have

$$W(u) = 1 + u^2 = \sum_{i=0}^{2} B_{i,2}(u) w_i = (1-u)^2 w_0 + 2u(1-u)w_1 + u^2 w_2$$



(a)



(b)

Figure 1.19. Representation of the unit circle. (a) $x(u) = (1-u^2)/(1+u^2)$ and $y(u) = (2u)/(1+u^2)$ for one quadrant; (b) the Bézier representation corresponding to Figure 1.19a ($w_0 = 1$, $w_1 = 1$, $w_2 = 2$).

Substituting $u = 0$ yields $w_0 = 1$, and $u = 1$ yields $w_2 = 2$. Finally, substituting $u = 1/2$ yields $5/4 = 1/4 w_0 + 1/2 w_1 + 1/4 w_2$, and using $w_0 = 1$ and $w_2 = 2$ yields $w_1 = 1$ (see Figure 1.19b).

Rational curves with coordinate functions in the form of Eq. (1.13) (one common denominator) have an elegant geometric interpretation which yields efficient processing and compact data storage. The idea is to use *homogeneous coordinates* to represent a rational curve in n-dimensional space as a polynomial curve in $(n+1)$-dimensional space (see [Robe65; Ries81; Patt85]). Let us start with a point in three-dimensional Euclidean space, $\mathbf{P} = (x, y, z)$. Then $\mathbf{P}$ is written as $\mathbf{P}^w = (wx, wy, wz, w) = (X, Y, Z, W)$ in four-dimensional space, $w \neq 0$. Now

$\mathbf{P}$ is obtained from $\mathbf{P}^w$ by dividing all coordinates by the fourth coordinate, W, i.e., by mapping $\mathbf{P}^w$ from the origin to the hyperplane $W = 1$ (see Figure 1.20 for the two-dimensional case, $\mathbf{P} = (x, y)$). This mapping, denoted by H, is a perspective map with center at the origin

$$\mathbf{P} = H\{\mathbf{P}^w\} = H\{(X, Y, Z, W)\} = \begin{cases} \left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}\right) & \text{if } W \neq 0 \\ \text{direction } (X, Y, Z) & \text{if } W = 0 \end{cases} \tag{1.16}$$

Notice that for arbitrary x, y, z, w_1, w_2, where $w_1 \neq w_2$

$$\begin{aligned} H\{\mathbf{P}^{w_1}\} &= H\{(w_1 x, w_1 y, w_1 z, w_1)\} = (x, y, z) \\ &= H\{(w_2 x, w_2 y, w_2 z, w_2)\} = H\{\mathbf{P}^{w_2}\} \end{aligned}$$

Now for a given set of control points, $\{\mathbf{P}_i\}$, and weights, $\{w_i\}$, construct the weighted control points, $\mathbf{P}_i^w = (w_i x_i, w_i y_i, w_i z_i, w_i)$. Then define the *nonrational* (polynomial) Bézier curve in four-dimensional space

$$\mathbf{C}^w(u) = \sum_{i=0}^{n} B_{i,n}(u)\, \mathbf{P}_i^w \tag{1.17}$$

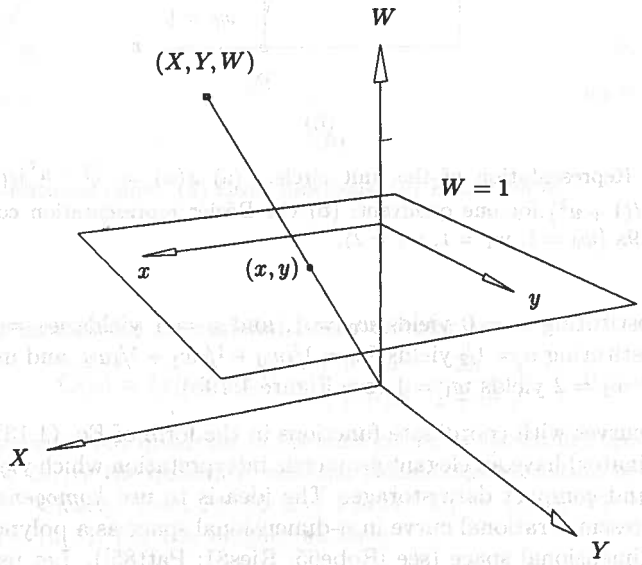


Figure 1.20. A representation of Euclidean points in homogeneous form.

Then, applying the perspective map, H, to $\mathbf{C}^w(u)$ yields the corresponding rational Bézier curve of Eq. (1.14) (see Figure 1.21), that is, writing out the coordinate functions of Eq. (1.17), we get

$$X(u) = \sum_{i=0}^{n} B_{i,n}(u) w_i x_i \qquad Y(u) = \sum_{i=0}^{n} B_{i,n}(u) w_i y_i$$

$$Z(u) = \sum_{i=0}^{n} B_{i,n}(u) w_i z_i \qquad W(u) = \sum_{i=0}^{n} B_{i,n}(u) w_i$$

Locating the curve in three-dimensional space yields

$$x(u) = \frac{X(u)}{W(u)} = \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i x_i}{\sum_{i=0}^{n} B_{i,n}(u) w_i}$$

$$y(u) = \frac{Y(u)}{W(u)} = \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i y_i}{\sum_{i=0}^{n} B_{i,n}(u) w_i}$$

$$z(u) = \frac{Z(u)}{W(u)} = \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i z_i}{\sum_{i=0}^{n} B_{i,n}(u) w_i}$$

Using vector notation, we get

$$\begin{aligned} \mathbf{C}(u) = (x(u), y(u), z(u)) &= \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i (x_i, y_i, z_i)}{\sum_{i=0}^{n} B_{i,n}(u) w_i} \\ &= \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i \mathbf{P}_i}{\sum_{i=0}^{n} B_{i,n}(u) w_i} \end{aligned} \tag{1.18}$$

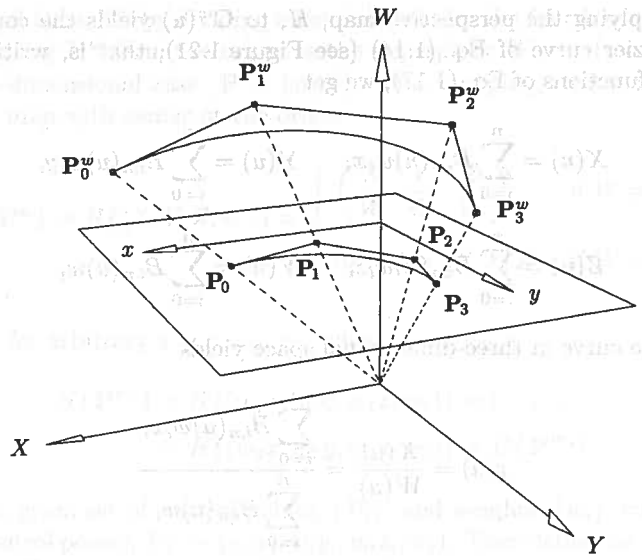


Figure 1.21. A geometric construction of a rational Bézier curve.

For algorithms in this book we primarily use the form given by Eq. (1.17), and an analogous form for rational B-spline curves. Thus, nonrational forms are processed in four-dimensional space, and the results are located in three-dimensional space using the map H. We refer interchangeably to either $\mathbf{C}^w(u)$ or $\mathbf{C}(u)$ as the rational Bézier (or B-spline) curve, although strictly speaking, $\mathbf{C}^w(u)$ is not a rational curve.

Examples

Ex1.13 Let us return to the circular arc of Figure 1.19b. We have $\mathbf{P}_0 = (1,0)$, $\mathbf{P}_1 = (1,1)$, $\mathbf{P}_2 = (0,1)$, and $w_0 = 1$, $w_1 = 1$, $w_2 = 2$. Hence, for Eq. (1.17) the three-dimensional control points are $\mathbf{P}_0^w = (1,0,1)$, $\mathbf{P}_1^w = (1,1,1)$, and $\mathbf{P}_2^w = (0,2,2)$. Then $\mathbf{C}^w(u) = (1-u)^2\,\mathbf{P}_0^w + 2u(1-u)\,\mathbf{P}_1^w + u^2\,\mathbf{P}_2^w$ is a parabolic arc (nonrational), which projects onto a circular arc on the $W = 1$ plane (see Figure 1.22).

Let u_0 be fixed. Since $\mathbf{C}^w(u)$ is a polynomial Bézier curve, we use the deCasteljau algorithm to compute $\mathbf{C}^w(u_0)$; subsequently, $\mathbf{C}(u_0) = H\{\mathbf{C}^w(u_0)\}$. Thus, we apply Eq. (1.12) to the $\mathbf{P}_i^w$

$$\mathbf{P}_{k,i}^w(u_0) = (1-u_0)\,\mathbf{P}_{k-1,i}^w + u_0\,\mathbf{P}_{k-1,i+1}^w \quad \text{for} \quad \begin{cases} k = 1,\ldots,n \\ i = 0,\ldots,n-k \end{cases} \tag{1.19}$$

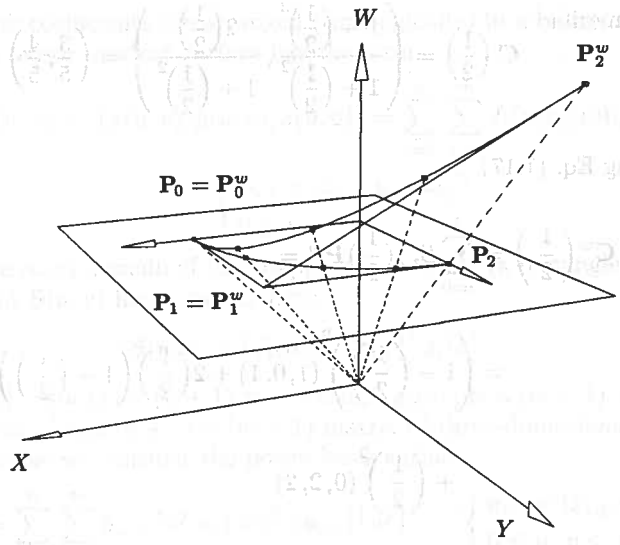


Figure 1.22. A homogeneous representation of a circular arc.

Ex1.14 Let us apply Eq. (1.19) to compute the point at $u = 1/2$ on the rational Bézier circular arc of Example 1.13. The arc is given by $\mathbf{C}^w(u) = (1-u)^2\,\mathbf{P}_0^w + 2u(1-u)\,\mathbf{P}_1^w + u^2\,\mathbf{P}_2^w$, where $\mathbf{P}_0^w = (1,0,1)$, $\mathbf{P}_1^w = (1,1,1)$, $\mathbf{P}_2^w = (0,2,2)$. The triangular set of generated points is shown in Table 1.4. Then $\mathbf{C}(1/2) = H\{\mathbf{C}^w(1/2)\} = H\{(3/4, 1, 5/4)\} = (3/5, 4/5)$.

Now let us compute the point using the other representations we have developed. Let

$$\mathbf{C}(u) = \left(\frac{(1-u^2)}{(1+u^2)}, \frac{(2u)}{(1+u^2)}\right)$$

Table 1.4. Generation of the point $\mathbf{C}^w(1/2)$ on the circular arc.

$(1,0,1)$		
	$\left(1, \frac{1}{2}, 1\right)$	
$(1,1,1)$		$\mathbf{C}^w\left(\frac{1}{2}\right)$
	$\left(\frac{3}{4}, 1, \frac{5}{4}\right)$	
	$\left(\frac{1}{2}, \frac{3}{2}, \frac{3}{2}\right)$	
$(0,2,2)$		

Then
$$\mathbf{C}\left(\tfrac{1}{2}\right) = \left(\frac{1-\left(\frac{1}{2}\right)^2}{1+\left(\frac{1}{2}\right)^2}, \frac{2\left(\frac{1}{2}\right)}{1+\left(\frac{1}{2}\right)^2}\right) = \left(\tfrac{3}{5}, \tfrac{4}{5}\right)$$

Using Eq. (1.17)

$$\mathbf{C}^w\left(\tfrac{1}{2}\right) = \sum_{i=0}^{2} B_{i,2}(\tfrac{1}{2})\mathbf{P}_i^w =$$

$$= \left(1-\left(\tfrac{1}{2}\right)\right)^2(1,0,1) + 2\left(\tfrac{1}{2}\right)\left(1-\left(\tfrac{1}{2}\right)\right)(1,1,1)$$

$$+ \left(\tfrac{1}{2}\right)^2(0,2,2)$$

$$= \tfrac{1}{4}(1,0,1) + \tfrac{1}{2}(1,1,1) + \tfrac{1}{4}(0,2,2) = \left(\tfrac{3}{4}, 1, \tfrac{5}{4}\right)$$

Projecting yields $(3/5, 4/5)$. Equations (1.18) and (1.15) yield the same result.

Finally, we note that $\mathbf{C}(1/2) = (3/5, 4/5)$ is not the midpoint of the circular arc in the first quadrant; i.e., the parameterization is not uniform (see Section 1.1). The point $(3/5, 4/5)$ is more than half the arc length from the starting point. This is intuitively correct, since by differentiating $\mathbf{C}(u)$ one can see that the starting speed is twice the end speed.

1.5 Tensor Product Surfaces

The curve $\mathbf{C}(u)$ is a vector-valued function of one parameter. It is a mapping (deformation) of a straight line segment into Euclidean three-dimensional space. A surface is a vector-valued function of two parameters, u and v, and represents a mapping of a region, $\mathcal{R}$, of the uv plane into Euclidean three-dimensional space. Thus it has the form $\mathbf{S}(u,v) = \big(x(u,v), y(u,v), z(u,v)\big)$, $(u,v) \in \mathcal{R}$. There are many schemes for representing surfaces (see [Hosc93; Pieg89a, 93] and the many references cited in [Pieg89a]). They differ in the coordinate functions used and the type of region $\mathcal{R}$. Probably the simplest method, and the one most widely used in geometric modeling applications, is the *tensor product* scheme. This is the method we use in the remainder of this book.

The tensor product method is basically a bidirectional curve scheme. It uses basis functions and geometric coefficients. The basis functions are bivariate functions of u and v, which are constructed as products of univariate basis functions.

The geometric coefficients are arranged (topologically) in a bidirectional, $n \times m$ net. Thus, a tensor product surface has the form

$$\mathbf{S}(u,v) = \big(x(u,v), y(u,v), z(u,v)\big) = \sum_{i=0}^{n}\sum_{j=0}^{m} f_i(u) g_j(v) \mathbf{b}_{i,j} \tag{1.20}$$

where
$$\begin{cases} \mathbf{b}_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j}) \\ 0 \le u, v \le 1 \end{cases}$$

Note that the (u,v) domain of this mapping is a square (a rectangle, in general). Note also that $\mathbf{S}(u,v)$ has a matrix form

$$\mathbf{S}(u,v) = [\,f_i(u)\,]^T\,[\,\mathbf{b}_{i,j}\,]\,[\,g_j(v)\,]$$

where $[\,f_i(u)\,]^T$ is a $(1) \times (n+1)$ row vector, $[\,g_j(v)\,]$ is a $(m+1) \times (1)$ column vector, and $[\,\mathbf{b}_{i,j}\,]$ is a $(n+1) \times (m+1)$ matrix of three-dimensional points.

As an example we consider the power basis surface

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{a}_{i,j} u^i v^j = [\,u^i\,]^T\,[\,\mathbf{a}_{i,j}\,]\,[\,v^j\,] \qquad \begin{cases} \mathbf{a}_{i,j} = (x_{i,j}, y_{i,j}, z_{i,j}) \\ 0 \le u, v \le 1 \end{cases} \tag{1.21}$$

We have $f_i(u) = u^i$ and $g_j(v) = v^j$, and the basis functions are the products, $\{u^i v^j\}$. If we fix $u = u_0$, then

$$\mathbf{C}_{u_0}(v) = \mathbf{S}(u_0, v) = \sum_{j=0}^{m}\left(\sum_{i=0}^{n} \mathbf{a}_{i,j}\, u_0^i\right) v^j = \sum_{j=0}^{m} \mathbf{b}_j(u_0)\, v^j \tag{1.22}$$

where
$$\mathbf{b}_j(u_0) = \sum_{i=0}^{n} \mathbf{a}_{i,j}\, u_0^i$$

is a power basis curve lying on the surface, $\mathbf{S}(u,v)$. Similarly, $\mathbf{C}_{v_0}(u)$ is a power basis curve lying on $\mathbf{S}(u,v)$; and the curves $\mathbf{C}_{u_0}(v)$ and $\mathbf{C}_{v_0}(u)$ intersect at the surface point, $\mathbf{S}(u_0, v_0)$. These curves are called *isoparametric curves* (or *isocurves*). $\mathbf{C}_{u_0}(v)$ is called a v *curve*, $\mathbf{C}_{v_0}(u)$ a u *curve* (see Figure 1.23).

Equation (1.21) can be written as

$$\begin{aligned} \mathbf{S}(u,v) = {} & \underbrace{\{\mathbf{a}_{0,0} + \mathbf{a}_{0,1}v + \mathbf{a}_{0,2}v^2 + \cdots + \mathbf{a}_{0,m}v^m\}}_{\mathbf{b}_0} \\ & + u\underbrace{\{\mathbf{a}_{1,0} + \mathbf{a}_{1,1}v + \mathbf{a}_{1,2}v^2 + \cdots + \mathbf{a}_{1,m}v^m\}}_{\mathbf{b}_1} \\ & + u^2\underbrace{\{\mathbf{a}_{2,0} + \mathbf{a}_{2,1}v + \mathbf{a}_{2,2}v^2 + \cdots + \mathbf{a}_{2,m}v^m\}}_{\mathbf{b}_2} \\ & \vdots \\ & + u^n\underbrace{\{\mathbf{a}_{n,0} + \mathbf{a}_{n,1}v + \mathbf{a}_{n,2}v^2 + \cdots + \mathbf{a}_{n,m}v^m\}}_{\mathbf{b}_n} \\ = {} & \mathbf{b}_0 + \mathbf{b}_1 u + \mathbf{b}_2 u^2 + \cdots + \mathbf{b}_n u^n \end{aligned}$$

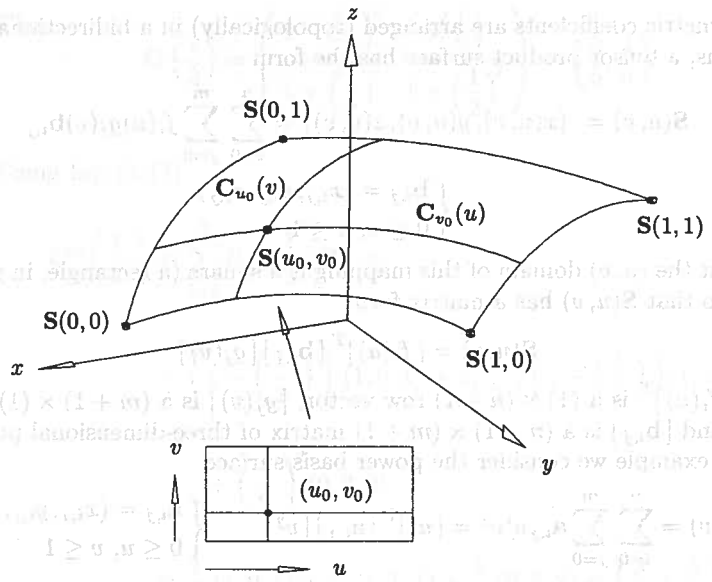


Figure 1.23. A tensor product surface showing isoparametric curves.

The terms in the braces are simple polynomials that can be evaluated by the Horner Algorithm (A1.1), yielding $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n$. Using the $\mathbf{b}$s and reapplying the algorithm, we obtain the point on the surface. Thus we have Algorithm A1.6.

```
ALGORITHM A1.6
  Horner2(a,n,m,u0,v0,S)
   { /*  Compute point on a power basis surface.  */
     /*  Input:  a,n,m,u0,v0  */
     /*  Output: S  */
    for (i=0; i<=n; i++)
      Horner1(a[i][],m,v0,b[i]); /* a[i][] is the ith row */
    Horner1(b,n,u0,S);
    }
```

Algorithm A1.6 is typical of the algorithms for tensor product surfaces. They can usually be obtained by extending from the curve algorithms, often by processing the n (or m) rows of coefficients (as curves) in one direction, then processing one or more rows in the other direction.

Differentiating Eq. (1.21), we obtain

$$\mathbf{S}_u(u,v) = \sum_{i=1}^{n}\sum_{j=0}^{m} i\,\mathbf{a}_{i,j}\,u^{i-1}v^j \qquad \mathbf{S}_v(u,v) = \sum_{i=0}^{n}\sum_{j=1}^{m} j\,\mathbf{a}_{i,j}\,u^i v^{j-1}$$

Notice that for fixed (u_0, v_0), $\mathbf{S}_u(u_0,v_0) = \mathbf{C}'_{v_0}(u_0)$ and $\mathbf{S}_v(u_0,v_0) = \mathbf{C}'_{u_0}(v_0)$. The normal vector, $\mathbf{N}$, is computed using Eq. (1.4).

Nonrational Bézier surfaces are obtained by taking a bidirectional net of control points and products of the univariate Bernstein polynomials

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)\mathbf{P}_{i,j} \qquad 0 \le u,v \le 1 \tag{1.23}$$

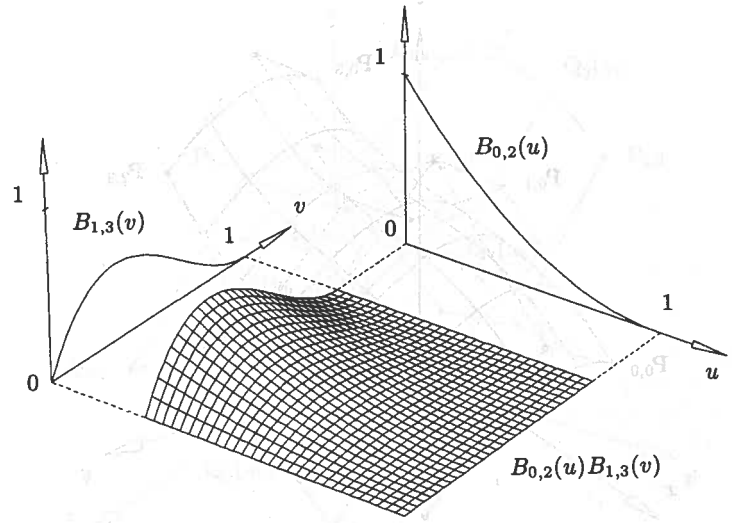
The basis function $B_{0,2}(u)B_{1,3}(v)$ is shown in Figure 1.24a, and Figure 1.24b shows a quadratic × cubic Bézier surface.

For fixed $u = u_0$

$$\begin{aligned}\mathbf{C}_{u_0}(v) = \mathbf{S}(u_0,v) &= \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u_0)B_{j,m}(v)\mathbf{P}_{i,j} \\ &= \sum_{j=0}^{m} B_{j,m}(v)\left(\sum_{i=0}^{n} B_{i,n}(u_0)\mathbf{P}_{i,j}\right) \\ &= \sum_{j=0}^{m} B_{j,m}(v)\,\mathbf{Q}_j(u_0)\end{aligned} \tag{1.24}$$

where $\mathbf{Q}_j(u_0) = \sum_{i=0}^{n} B_{i,n}(u_0)\mathbf{P}_{i,j} \qquad j = 0,\ldots,m$

is a Bézier curve lying on the surface. Analogously, $\mathbf{C}_{v_0}(u) = \sum_{i=0}^{n} B_{i,n}(u)\,\mathbf{Q}_i(v_0)$ is a Bézier u isocurve lying on the surface.



(a)

Figure 1.24. (a) The Bézier tensor product basis function, $B_{0,2}(u)B_{1,3}(v)$; (b) a quadratic × cubic Bézier surface.

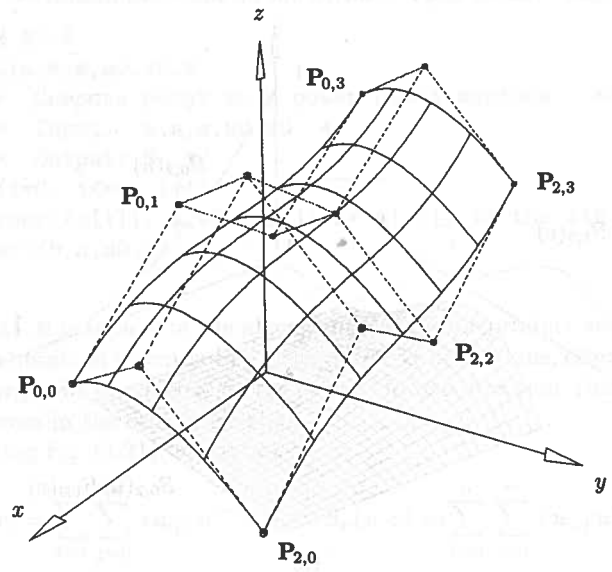
As is the case for curves, because of their excellent properties Bézier surfaces are better suited for geometric modeling applications than power basis surfaces. In particular,

- nonnegativity: $B_{i,n}(u)B_{j,m}(v) \geq 0$ for all i, j, u, v;
- partition of unity: $\sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v) = 1$ for all u and v;
- $\mathbf{S}(u, v)$ is contained in the convex hull of its control points;
- transformation invariance;
- the surface interpolates the four corner control points;
- when triangulated, the control net forms a planar polyhedral approximation to the surface.

It is interesting to note that there is no known variation diminishing property for Bézier surfaces (see [Prau92]).

The deCasteljau algorithm (`A1.5`) is also easily extended to compute points on a Bézier surface. Refer to Eq. (1.24) and Figure 1.25. Let (u_0, v_0) be fixed. For fixed j_0, $\mathbf{Q}_{j_0}(u_0) = \sum_{i=0}^{n} B_{i,n}(u_0)\mathbf{P}_{i,j_0}$ is the point obtained by applying the deCasteljau algorithm to the j_0 row of control points, i.e., to $\{\mathbf{P}_{i,j_0}\}$, $i = 0, \ldots, n$. Therefore, applying the deCasteljau Algorithm $(m+1)$ times yields $\mathbf{C}_{u_0}(v)$; and applying it once more to $\mathbf{C}_{u_0}(v)$ at $v = v_0$ yields $\mathbf{C}_{u_0}(v_0) = \mathbf{S}(u_0, v_0)$. This process requires

$$\frac{n(n+1)(m+1)}{2} + \frac{m(m+1)}{2} \tag{1.25}$$



(b)

Figure 1.24. (*Continued.*)

linear interpolations (see Exercise 1.21). By symmetry, we can compute $\mathbf{C}_{v_0}(u)$ first ($n+1$ applications of deCasteljau) and then compute $\mathbf{C}_{v_0}(u_0) = \mathbf{S}(u_0, v_0)$. This requires

$$\frac{m(m+1)(n+1)}{2} + \frac{n(n+1)}{2} \tag{1.26}$$

linear interpolations. Thus, if $n > m$ compute $\mathbf{C}_{v_0}(u)$ first, then $\mathbf{C}_{v_0}(u_0)$; otherwise, compute $\mathbf{C}_{u_0}(v)$ first, then $\mathbf{C}_{u_0}(v_0)$.

```
ALGORITHM A1.7
 deCasteljau2(P,n,m,u0,v0,S)
   { /*  Compute a point on a Bézier surface */
     /*  by the deCasteljau.  */
     /*  Input:  P,n,m,u0,v0  */
     /*  Output: S
   if (n <= m)
     {
     for (j=0; j<=m; j++) /* P[j][] is jth row */
       deCasteljau1(P[j][],n,u0,Q[j]);
     deCasteljau1(Q,m,v0,S);
     }
   else
     {
     for (i=0; i<=n; i++)
```

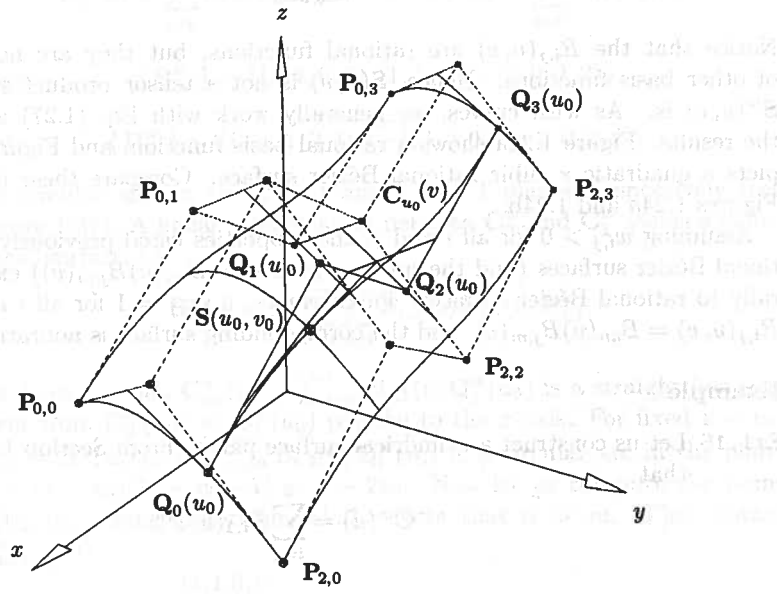


Figure 1.25. The deCasteljau algorithm for a Bézier surface.

```
        deCasteljau1(P[][i],m,v0,Q[i]);
      deCasteljau1(Q,n,u0,S);
    }
  }
```

We define a *rational Bézier surface* to be the perspective projection of a four-dimensional polynomial Bézier surface (see [Pieg86; Fari89])

$$\mathbf{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)\mathbf{P}^w_{i,j} \tag{1.27}$$

and

$$\mathbf{S}(u,v) = H\{\mathbf{S}^w(u,v)\} = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)w_{i,j}}$$

$$= \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}(u,v)\mathbf{P}_{i,j} \tag{1.28}$$

where

$$R_{i,j}(u,v) = \frac{B_{i,n}(u)B_{j,m}(v)w_{i,j}}{\sum_{r=0}^{n}\sum_{s=0}^{m} B_{r,n}(u)B_{s,m}(v)w_{r,s}}$$

Notice that the $R_{i,j}(u,v)$ are rational functions, but they are not products of other basis functions. Hence, $\mathbf{S}(u,v)$ is not a tensor product surface, but $\mathbf{S}^w(u,v)$ is. As with curves, we generally work with Eq. (1.27) and project the results. Figure 1.26a shows a rational basis function, and Figure 1.26b depicts a quadratic × cubic rational Bézier surface. Compare these figures with Figures 1.24a and 1.24b.

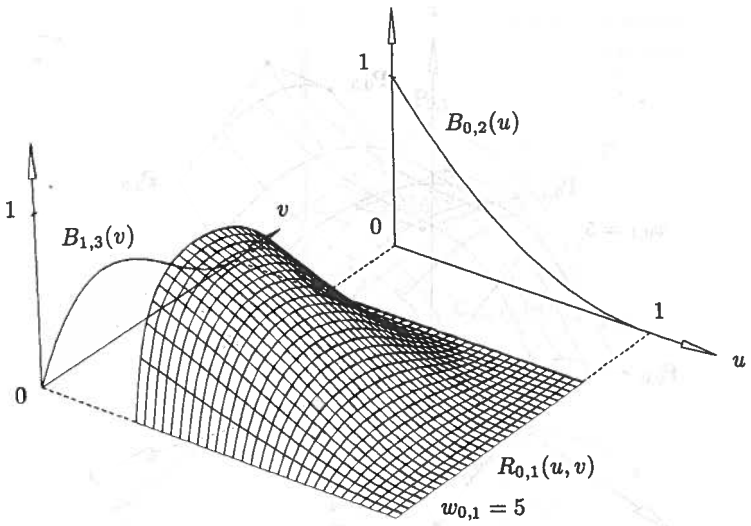
Assuming $w_{i,j} > 0$ for all i and j, the properties listed previously for nonrational Bézier surfaces (and the product functions $B_{i,n}(u)B_{j,m}(v)$) extend naturally to rational Bézier surfaces. Furthermore, if $w_{i,j} = 1$ for all i and j, then $R_{i,j}(u,v) = B_{i,n}(u)B_{j,m}(v)$, and the corresponding surface is nonrational.

Example

Ex1.15 Let us construct a cylindrical surface patch. From Section 1.4 we know that

$$\mathbf{C}^w(u) = \sum_{i=0}^{2} B_{i,2}(u)\mathbf{P}^w_i$$

for $\{\mathbf{P}^w_i\} = \{(0,1,0,1),(0,1,1,1),(0,0,2,2)\}$, is a circular arc in the yz plane. Using translation (P1.14, Section 1.4)



(a)

Figure 1.26. (a) The rational basis function $R_{0,1}(u,v)$ (with $w_{0,1} = 5$ and all other weights equal to zero); (b) a quadratic × cubic rational Bézier surface.

$$\mathbf{C}^w_0(u) = \sum_{i=0}^{2} B_{i,2}(u)\mathbf{P}^w_{i,0} \quad \text{and} \quad \mathbf{C}^w_1(u) = \sum_{i=0}^{2} B_{i,2}(u)\mathbf{P}^w_{i,1}$$

where $\{\mathbf{P}^w_{i,0}\} = \{(1,1,0,1),(1,1,1,1),(2,0,2,2)\}$

and $\{\mathbf{P}^w_{i,1}\} = \{(-1,1,0,1),(-1,1,1,1),(-2,0,2,2)\}$

are circular arcs in the $x = 1$ and $x = -1$ planes, respectively (see Figure 1.27). A linear interpolation between $\mathbf{C}^w_0$ and $\mathbf{C}^w_1$ yields a cylindrical surface, i.e.

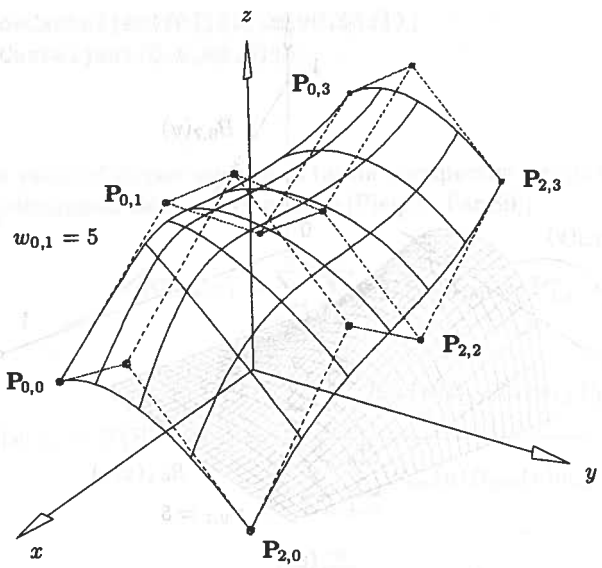
$$\mathbf{S}^w(u,v) = \sum_{i=0}^{2}\sum_{j=0}^{1} B_{i,2}(u)B_{j,1}(v)\mathbf{P}^w_{i,j}$$

For fixed $u = u_0$, $\mathbf{C}^w_{u_0}(v) = \sum_{j=0}^{1} B_{j,1}(v)\mathbf{Q}^w_j(u_0)$ is a straight line segment from $\mathbf{C}^w_0(u_0)$ to $\mathbf{C}^w_1(u_0)$ parallel to the x-axis. For fixed $v = v_0$, $\mathbf{C}^w_{v_0} = \mathbf{S}^w(u,v_0) = \sum_{i=0}^{2} B_{i,2}(u)\mathbf{Q}^w_i(v_0)$ is a circular arc in the plane $x = (1-v_0)(1) + v_0(-1) = 1 - 2v_0$. Now let us compute the point $\mathbf{S}(1/2, 1/2)$, using Algorithm A1.7. Note that $n > m$. First obtain $\mathbf{C}^w_{v_0=1/2}(u)$

(1,1,0,1)

(0,1,0,1) = $\mathbf{Q}^w_0(v_0)$

(-1,1,0,1)



(b)

Figure 1.26. (b) (*Continued.*)

$$\begin{matrix}(1,1,1,1)\\(-1,1,1,1)\end{matrix} \qquad (0,1,1,1) = \mathbf{Q}_1^w(v_0)$$

$$\begin{matrix}(2,0,2,2)\\(-2,0,2,2)\end{matrix} \qquad (0,0,2,2) = \mathbf{Q}_2^w(v_0)$$

Now $\mathbf{C}^w_{v_0=1/2}(u) = \sum_{i=0}^{2} B_{i,2}(u)\mathbf{Q}_i^w(v_0)$ is the circular arc in the yz plane. Then

$$\begin{matrix}(0,1,0,1)\\ \\(0,1,1,1)\\ \\(0,0,2,2)\end{matrix} \qquad \begin{matrix}\left(0,1,\frac{1}{2},1\right)\\ \\ \left(0,\frac{3}{4},1,\frac{5}{4}\right) = \mathbf{S}^w\left(\frac{1}{2},\frac{1}{2}\right)\\ \\ \left(0,\frac{1}{2},\frac{3}{2},\frac{3}{2}\right)\end{matrix}$$

And projecting yields

$$\mathbf{S}\left(\frac{1}{2},\frac{1}{2}\right) = H\left\{\left(0,\frac{3}{4},1,\frac{5}{4}\right)\right\} = \left(0,\frac{3}{5},\frac{4}{5}\right)$$

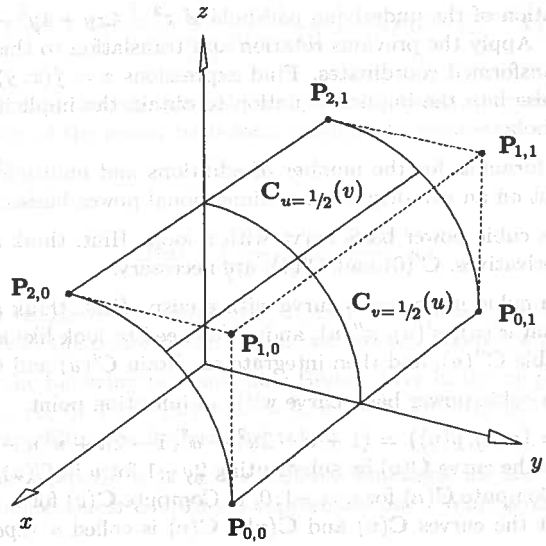


Figure 1.27. A cylindrical surface patch as a rational Bézier surface.

Exercises

1.1. Consider the two parametric representations of the circular arc given by Eqs. (1.1) and (1.2). Using Eq. (1.1), compute the curve point at $u = \pi/4$ and, using Eq. (1.2), the point at $t = 1/2$. Explain the results.

1.2. Compute the acceleration vector, $\mathbf{C}''(u)$, for Eq. (1.1). Explain the result.

1.3. Using trigonometric functions, give a parametric definition of the bounded surface of

- a right circular cone, with apex at the origin and axis of symmetry along the z-axis;
- the cone is opening upward, and is bounded above at $z = 2$ by the circle with radius = 1.

Modify Eq. (1.2) to get another representation of the same cone. Compute the first partial derivatives, $\mathbf{S}_u$ and $\mathbf{S}_v$, of the trigonometric representation. What are the values of these derivatives at the apex of the cone?

1.4. Consider the parabolic arc $\mathbf{C}(u) = (x(u), y(u)) = (-1 - u + 2u^2, -2u + u^2)$, $0 \le u \le 1$. Sketch this curve. The curve is rotated and translated by applying the transformations to the functions $x(u)$ and $y(u)$. Apply the two transformations

(1) 90° rotation about the origin. The rotation matrix (applied from the left) is

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

(2) translation with the vector $(-1, -1)$.

The implicit equation of the underlying parabola is $x^2 - 4xy + 4y^2 - 4x - y - 5 = 0$. Sketch this curve. Apply the previous rotation and translation to this equation. Hint: let $\bar{x}, \bar{y}$ be the transformed coordinates. Find expressions $x = f(\bar{x}, \bar{y})$ and $y = g(\bar{x}, \bar{y})$ and substitute these into the implicit equation to obtain the implicit equation of the transformed parabola.

1.5. Determine formulas for the number of additions and multiplications necessary to compute a point on an nth-degree three-dimensional power basis curve.

1.6. Construct a cubic power basis curve with a loop. Hint: think about what endpoints and end derivatives, $\mathbf{C}'(0)$ and $\mathbf{C}'(1)$, are necessary.

1.7. Construct a cubic power basis curve with a cusp. Hint: think about $\mathbf{C}'(u)$ and $\mathbf{C}''(u)$. Sketch what $x'(u)$, $y'(u)$, $x''(u)$, and $y''(u)$ need to look like as functions of u. Determine a suitable $\mathbf{C}''(u)$, and then integrate to obtain $\mathbf{C}'(u)$ and $\mathbf{C}(u)$.

1.8. Construct a cubic power basis curve with an inflection point.

1.9. Let $\mathbf{C}(u) = (x(u), y(u)) = (1 + u - 2u^2 + u^3, 1 - 2u + u^3)$, $-1 \le u \le 1$. Let $u = 2v - 1$. Derive the curve $\mathbf{C}(v)$ by substituting $2v - 1$ for u in $\mathbf{C}(u)$. What degree is the curve $\mathbf{C}(v)$? Compute $\mathbf{C}(u)$ for $u = -1, 0, 1$. Compute $\mathbf{C}(v)$ for $v = 0, 1/2, 1$. What can you say about the curves $\mathbf{C}(u)$ and $\mathbf{C}(v)$? $\mathbf{C}(v)$ is called a *reparameterization* of $\mathbf{C}(u)$.

1.10. Check the property **P1.7** of the Bernstein polynomials for the cases $n = 2$ and $n = 3$.

1.11. It is sometimes necessary to reverse a curve, i.e., given $\mathbf{C}_1(u)$, $0 \le u \le 1$, produce $\mathbf{C}_2(v)$, $0 \le v \le 1$, such that the two curves are the same geometrically, but $\mathbf{C}_1(0) = \mathbf{C}_2(1)$ and $\mathbf{C}_1(1) = \mathbf{C}_2(0)$. How would you do this using the Bézier form? The power basis form?

1.12. Consider the xy planar cubic Bézier curve given by the control points $\mathbf{P}_0 = (0, 6)$, $\mathbf{P}_1 = (3, 6)$, $\mathbf{P}_2 = (6, 3)$, $\mathbf{P}_3 = (6, 0)$. Compute the point $\mathbf{C}(1/3)$ using the deCasteljau algorithm. Compute the same point by using Eqs. (1.7) and (1.8) directly, i.e., evaluate the basis functions at $u = 1/3$ and multiply by the appropriate control points.

1.13. Determine formulas for the number of additions and multiplications necessary to compute a point on an nth-degree three-dimensional Bézier curve using the deCasteljau algorithm, **A1.5**, and algorithm **A1.4**. Compare the results with Exercise 1.5 (the Horner algorithm).

1.14. For given n, row vector $[B_{0,n}(u), \ldots, B_{n,n}(u)]$ can be written as $[1, u, \ldots, u^n]M$, where M is an $(n+1) \times (n+1)$ matrix. Thus, a Bézier curve can be written in matrix form, $\mathbf{C}(u) = [u^i]^T M[\mathbf{P}_i]$. Compute the matrices M for $n = 1, 2, 3$. Notice that setting $[\mathbf{a}_i] = M[\mathbf{P}_i]$ yields the conversion of a Bézier curve to power basis form. Assuming $0 \le u \le 1$, $[\mathbf{P}_i] = M^{-1}[\mathbf{a}_i]$ gives the conversion from power basis to Bézier form.

1.15. Compare this with Exercise 1.9. It is also possible to define a Bézier curve on a parameter interval other than $[0, 1]$. This is equivalent to reparameterizing a Bézier curve. Let

$$\mathbf{C}(u) = \sum_{i=0}^{n} B_{i,n}(u)\mathbf{P}_i \qquad u \in [0, 1]$$

Let $v \in [a, b]$. Then $u = (v - a)/(b - a)$. Substitute this equation into Eq. (1.8) and derive this expression for the reparameterized curve

$$\mathbf{C}(v) = \frac{1}{(b-a)^n} \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} (v-a)^i (b-v)^{n-i} \mathbf{P}_i$$

It is interesting to note that the control points do not change, only the basis functions. Reparameterization of the power basis form changes the geometric coefficients but not the basis functions.

1.16. Consider the circle

$$\mathbf{C}(u) = \left(\frac{1 - u^2}{1 + u^2}, \frac{2u}{1 + u^2} \right)$$

Determine which ranges of the parameter u yield which quadrants of the circle. Do these equations yield the entire circle? What can you say about the parameterization?

1.17. Consider the following rational cubic Bézier curve in the xy plane: $\mathbf{P}_0 = (0, 6)$, $\mathbf{P}_1 = (3, 6)$, $\mathbf{P}_2 = (6, 3)$, $\mathbf{P}_3 = (6, 0)$, $w_0 = 4$, $w_1 = 1$, $w_2 = 1$, $w_3 = 4$. Compute the point $\mathbf{C}(2/3)$ by expanding the deCasteljau table.

1.18. What characteristic is it of the rational functions we are using that allows us to use the homogeneous coordinate representation? Why is this representation advantageous?

1.19. Find the rational Bézier representation of the circular arc in the second quadrant, i.e., determine the $\mathbf{P}_i$ and w_i. Hint: use symmetry and check your result by showing that $(x(u))^2 + (y(u))^2 = 1$ for all $u \in [0, 1]$.

1.20. The circular arc in the first quadrant is also given by the equation

$$\mathbf{C}(u) = \left(\frac{1 + (\sqrt{2} - 2)u + (1 - \sqrt{2})u^2}{1 + (\sqrt{2} - 2)u + (2 - \sqrt{2})u^2}, \frac{\frac{\sqrt{2}}{2} u\Big((\sqrt{2} - 2)u + 2\Big)}{1 + (\sqrt{2} - 2)u + (2 - \sqrt{2})u^2} \right)$$

Determine the rational Bézier representation corresponding to these equations. Hint: the $\mathbf{P}_i$ must be the same as before – $(1, 0)$, $(1, 1)$, $(0, 1)$; Why? Compute the weights w_i by equating polynomials and substituting $u = 0, 1/2, 1$, as done previously. Compute the point $\mathbf{C}(1/2)$, using any method. What is interesting about $\mathbf{C}(1/2)$?

1.21. Derive Eqs. (1.25) and (1.26). Hint: use the formula $1 + 2 + \cdots + n = n(n+1)/2$.

1.22. For the cylindrical surface example (**Ex1.15**) compute the control points $\mathbf{Q}_j^w(u_0)$ for the isocurve $\mathbf{C}^w_{u_0 = 1/3}(v)$.

1.23. Let $n = 3$ and $m = 2$. Consider the nonrational Bézier surface defined by the control net

$$\{\mathbf{P}_{i,0}\} = \{(0, 0, 0), (3, 0, 3), (6, 0, 3), (9, 0, 0)\}$$

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

$$\{\mathbf{P}_{i,2}\} = \{(0, 4, 0), (3, 4, 3), (6, 4, 3), (9, 4, 0)\}$$

a. sketch this surface;

b. use the deCasteljau algorithm to compute the surface point $\mathbf{S}(1/3, 1/2)$;

c. fix $u_0 = 1/2$ and extract the Bézier representation (control points) of the curve $\mathbf{C}_{u_0 = 1/2}(v)$.

1.24. Let

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)\,\mathbf{P}_{i,j}$$

and assume that $\mathbf{P}_{0,0} = \mathbf{P}_{1,0} = \cdots = \mathbf{P}_{n,0}$. How does this affect $\mathbf{S}(u,v)$, the derivatives $\mathbf{S}_u(u,v)$ and $\mathbf{S}_v(u,v)$, and the curves $\mathbf{C}_{v_0}(u)$? Assume that $\mathbf{P}_{i,0} = (1,0,0)$ for $i = 0,1,2$ in Example **Ex1.15**, with $w_{0,0} = 1$, $w_{1,0} = 1$, and $w_{2,0} = 2$. What type of surface do you get?

1.25. The prerequisite for this problem is Exercise 1.14. The rational Bézier surface (Eq. [1.27]) has a matrix form

$$\mathbf{S}^w(u,v) = [\,B_{i,n}(u)\,]^T\,[\,\mathbf{P}^w_{i,j}\,]\,[\,B_{j,m}(v)\,] = [\,u^i\,]^T\,M_n\,[\,\mathbf{P}^w_{i,j}\,]\,M_m^T\,[\,v^j\,]$$

where $[\,u^i\,]^T$ and $[\,v^j\,]$ are vectors, M_n is an $(n+1)\times(n+1)$ matrix, M_m^T is a $(m+1)\times(m+1)$ matrix, and $[\,\mathbf{P}^w_{i,j}\,]$ is an $(n+1)\times(m+1)$ matrix of four-dimensional points. Write this form down explicitly for the cylindrical surface example, **Ex1.15**. Using this matrix form, compute the point $\mathbf{S}^w(1/2, 1/2)$, and then project to obtain $\mathbf{S}(1/2, 1/2)$. There is no direct matrix form for $\mathbf{S}(u,v)$; why not?

CHAPTER

TWO

B-Spline Basis Functions

2.1 Introduction

Curves consisting of just one polynomial or rational segment are often inadequate. Their shortcomings are:

- a high degree is required in order to satisfy a large number of constraints; e.g., $(n-1)$-degree is needed to pass a polynomial Bézier curve through n data points. However, high degree curves are inefficient to process and are numerically unstable;
- a high degree is required to accurately fit some complex shapes;
- single-segment curves (surfaces) are not well-suited to interactive shape design; although Bézier curves can be shaped by means of their control points (and weights), the control is not sufficiently local.

The solution is to use curves (surfaces) which are *piecewise polynomial*, or *piecewise rational*. Figure 2.1 shows a curve, $\mathbf{C}(u)$, consisting of m $(= 3)$ nth-degree polynomial *segments*. $\mathbf{C}(u)$ is defined on $u \in [0,1]$. The parameter values $u_0 = 0 < u_1 < u_2 < u_3 = 1$ are called *breakpoints*. They map into the endpoints of the three polynomial segments. We denote the segments by $\mathbf{C}_i(u)$, $1 \le i \le m$. The segments are constructed so that they join with some level of continuity (not necessarily the same at every breakpoint). Let $\mathbf{C}_i^{(j)}$ denote the jth derivative of $\mathbf{C}_i$. $\mathbf{C}(u)$ is said to be C^k continuous at the breakpoint u_i if $\mathbf{C}_i^{(j)}(u_i) = \mathbf{C}_{i+1}^{(j)}(u_i)$ for all $0 \le j \le k$.

Any of the standard polynomial forms can be used to represent $\mathbf{C}_i(u)$. Figure 2.2 shows the curve of Figure 2.1 with the three segments in cubic Bézier form. $\mathbf{P}_i^j$ denotes the ith control point of the jth segment.

If the degree equals three and the breakpoints $U = \{u_0, u_1, u_2, u_3\}$ remain fixed, and if we allow the twelve control points, $\mathbf{P}_i^j$, to vary arbitrarily, we obtain the vector space, $\mathcal{V}$, consisting of all piecewise cubic polynomial curves

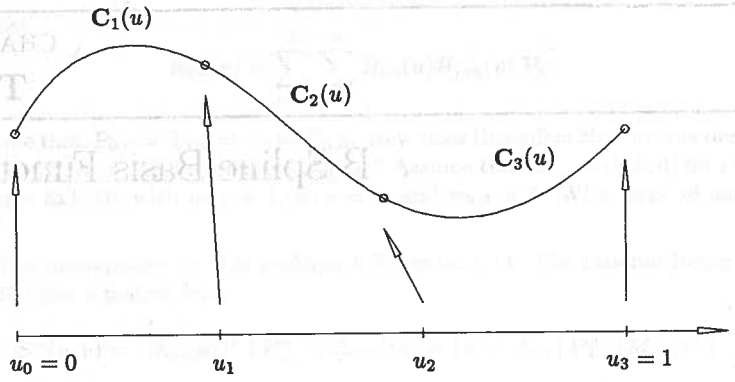


Figure 2.1. A piecewise cubic polynomial curve with three segments.

on U. $\mathcal{V}$ has dimension twelve, and a curve in $\mathcal{V}$ may be discontinuous at u_1 or u_2. Now suppose we specify (as in Figure 2.2) that $\mathbf{P}_3^1 = \mathbf{P}_0^2$ and $\mathbf{P}_3^2 = \mathbf{P}_0^3$. This gives rise to $\mathcal{V}^0$, the vector space of all piecewise cubic polynomial curves on U which are at least C^0 continuous everywhere. $\mathcal{V}^0$ has dimension ten, and $\mathcal{V}^0 \subset \mathcal{V}$.

Imposing C^1 continuity is a bit more involved. Let us consider $u = u_1$. Assume that $\mathbf{P}_3^1 = \mathbf{P}_0^2$. Let

$$v = \frac{u - u_0}{u_1 - u_0} \quad \text{and} \quad w = \frac{u - u_1}{u_2 - u_1}$$

be local parameters on the intervals $[u_0, u_1]$ and $[u_1, u_2]$, respectively. Then $0 \le v, w \le 1$. C^1 continuity at u_1 implies

$$\frac{1}{u_1 - u_0}\mathbf{C}_1^{(1)}(v = 1) = \mathbf{C}_1^{(1)}(u_1) = \mathbf{C}_2^{(1)}(u_1) = \frac{1}{u_2 - u_1}\mathbf{C}_2^{(1)}(w = 0)$$

and from Eq. (1.10) it follows that

$$\frac{3}{u_1 - u_0}(\mathbf{P}_3^1 - \mathbf{P}_2^1) = \frac{3}{u_2 - u_1}(\mathbf{P}_1^2 - \mathbf{P}_0^2)$$

Thus

$$\mathbf{P}_3^1 = \frac{(u_2 - u_1)\mathbf{P}_2^1 + (u_1 - u_0)\mathbf{P}_1^2}{u_2 - u_0} \tag{2.1}$$

Equation (2.1) says that $\mathbf{P}_3^1$ and $\mathbf{P}_3^2$ can be written in terms of $\mathbf{P}_2^1, \mathbf{P}_1^2$ and $\mathbf{P}_2^2, \mathbf{P}_1^3$, respectively. Hence, $\mathcal{V}^1$, the vector space of all C^1 continuous piecewise cubic polynomial curves on U, has dimension eight, and $\mathcal{V}^1 \subset \mathcal{V}^0 \subset \mathcal{V}$.

This makes it clear that storing and manipulating the individual polynomial segments of a piecewise polynomial curve is not the ideal method for handling

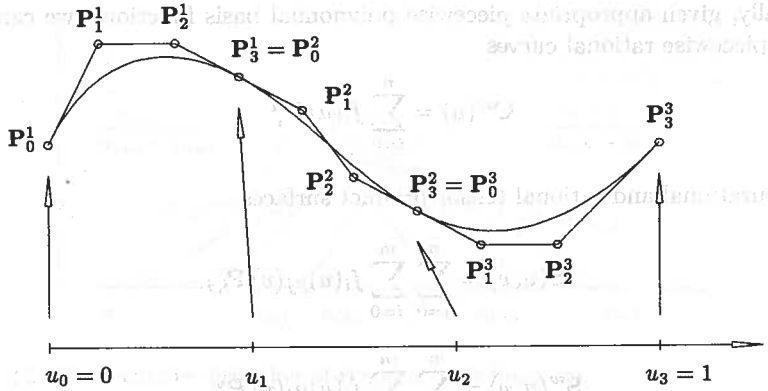


Figure 2.2. The curve of Figure 2.1 shown with the polynomial segments represented in Bézier form.

such curves. First, redundant data must be stored: twelve coefficients, where only eight are required for C^1 continuous cubic curves, and only six for C^2 continuous cubic curves. Second, for the Bézier form the continuity of $\mathbf{C}(u)$ depends on the positions of the control points, hence there is little flexibility in positioning control points while maintaining continuity. If a designer wants C^1 continuity and is satisfied with the segments $\mathbf{C}_1(u)$ and $\mathbf{C}_3(u)$, but wants to modify the shape of $\mathbf{C}_2(u)$, he is out of luck: none of $\mathbf{C}_2(u)$'s control points can be modified. Third, determining the continuity of a curve requires computation (such as Eq. [2.1]).

We want a curve representation of the form

$$\mathbf{C}(u) = \sum_{i=0}^{n} f_i(u)\,\mathbf{P}_i \tag{2.2}$$

where the $\mathbf{P}_i$ are *control points*, and the $\{f_i(u),\ i = 0, \ldots, n\}$ are *piecewise polynomial functions* forming a basis for the vector space of all piecewise polynomial functions of the desired degree and continuity (for a fixed breakpoint sequence, $U = \{u_i\},\ 0 \le i \le m$). Note that continuity is determined by the basis functions, hence the control points can be modified without altering the curve's continuity. Furthermore, the $\{f_i\}$ should have the 'usual' nice analytic properties, e.g. those listed in Section 1.3. This ensures that the curves defined by Eq. (2.2) have nice geometric properties similar to Bézier curves, e.g., convex hull, variation diminishing, transformation invariance. Another important property that we seek in our basis functions is that of *local support*; this implies that each $f_i(u)$ is nonzero only on a limited number of subintervals, not the entire domain, $[u_0, u_m]$. Since $\mathbf{P}_i$ is multiplied by $f_i(u)$, moving $\mathbf{P}_i$ affects curve shape only on the subintervals where $f_i(u)$ is nonzero.

Finally, given appropriate piecewise polynomial basis functions, we can construct piecewise rational curves

$$\mathbf{C}^w(u) = \sum_{i=0}^{n} f_i(u)\,\mathbf{P}_i^w \tag{2.3}$$

and nonrational and rational tensor product surfaces

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} f_i(u)g_j(v)\,\mathbf{P}_{i,j}$$

$$\mathbf{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} f_i(u)g_j(v)\,\mathbf{P}_{i,j}^w \tag{2.4}$$

For the remainder of this chapter we study the so-called B-spline basis functions. In Chapters 3 and 4 we combine these functions with three-dimensional and four-dimensional control points to obtain nonrational and rational curves and surfaces, respectively.

2.2 Definition and Properties of B-spline Basis Functions

There are a number of ways to define the B-spline basis functions and to prove their important properties, e.g., by divided differences of truncated power functions [Curr47; Scho46], by blossoming [Rams87], and by a recurrence formula due to deBoor, Cox, and Mansfield [Cox72; DeBo72, 78]. We use the recurrence formula, since it is the most useful for computer implementation.

Let $U = \{u_0, \ldots, u_m\}$ be a nondecreasing sequence of real numbers, i.e., $u_i \le u_{i+1}$, $i = 0, \ldots, m-1$. The u_i are called *knots*, and U is the *knot vector*. The ith B-spline basis function of p-degree (*order* $p+1$), denoted by $N_{i,p}(u)$, is defined as

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \le u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.5}$$

Note that

- $N_{i,0}(u)$ is a step function, equal to zero everywhere except on the half-open interval $u \in [u_i, u_{i+1})$;
- for $p > 0$, $N_{i,p}(u)$ is a linear combination of two $(p-1)$-degree basis functions (Figure 2.3);

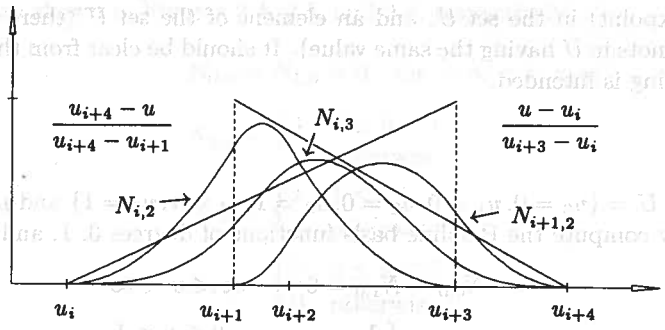


Figure 2.3. The recursive definition of B-spline basis functions.

- computation of a set of basis functions requires specification of a knot vector, U, and the degree, p;
- Equation (2.5) can yield the quotient $0/0$ (see examples later); we define this quotient to be zero;
- the $N_{i,p}(u)$ are piecewise polynomials, defined on the entire real line; generally only the interval $[u_0, u_m]$ is of interest;
- the half-open interval, $[u_i, u_{i+1})$, is called the *ith knot span*; it can have zero length, since knots need not be distinct;
- the computation of the pth-degree functions generates a truncated triangular table

$$\begin{array}{ccccc}
N_{0,0} & & & & \\
 & N_{0,1} & & & \\
N_{1,0} & & N_{0,2} & & \\
 & N_{1,1} & & N_{0,3} & \\
N_{2,0} & & N_{1,2} & & \\
 & N_{2,1} & & N_{1,3} & \\
N_{3,0} & & N_{2,2} & \vdots & \\
 & N_{3,1} & \vdots & & \\
N_{4,0} & \vdots & & & \\
\vdots & & & &
\end{array}$$

For brevity we often write $N_{i,p}$ instead of $N_{i,p}(u)$.

A word about terminology. In Section 2.1 we used the term breakpoint and required $u_i < u_{i+1}$ for all i. In the remainder of this book we use the term knot and assume $u_i \le u_{i+1}$. The breakpoints correspond to the set of *distinct* knot values, and the knot spans of nonzero length define the individual polynomial segments. Hence, we use the word knot with two different meanings: a distinct

value (breakpoint) in the set U, and an element of the set U (there can exist additional knots in U having the same value). It should be clear from the context which meaning is intended.

Examples

Ex2.1 Let $U = \{u_0 = 0, u_1 = 0, u_2 = 0, u_3 = 1, u_4 = 1, u_5 = 1\}$ and $p = 2$. We now compute the B-spline basis functions of degrees 0, 1, and 2

$$N_{0,0} = N_{1,0} = 0 \qquad -\infty < u < \infty$$

$$N_{2,0} = \begin{cases} 1 & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{3,0} = N_{4,0} = 0 \qquad -\infty < u < \infty$$

$$N_{0,1} = \frac{u-0}{0-0}N_{0,0} + \frac{0-u}{0-0}N_{1,0} = 0 \qquad -\infty < u < \infty$$

$$N_{1,1} = \frac{u-0}{0-0}N_{1,0} + \frac{1-u}{1-0}N_{2,0} = \begin{cases} 1-u & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{2,1} = \frac{u-0}{1-0}N_{2,0} + \frac{1-u}{1-1}N_{3,0} = \begin{cases} u & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{3,1} = \frac{u-1}{1-1}N_{3,0} + \frac{1-u}{1-1}N_{4,0} = 0 \qquad -\infty < u < \infty$$

$$N_{0,2} = \frac{u-0}{0-0}N_{0,1} + \frac{1-u}{1-0}N_{1,1} = \begin{cases} (1-u)^2 & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{1,2} = \frac{u-0}{1-0}N_{1,1} + \frac{1-u}{1-0}N_{2,1} = \begin{cases} 2u(1-u) & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{2,2} = \frac{u-0}{1-0}N_{2,1} + \frac{1-u}{1-1}N_{3,1} = \begin{cases} u^2 & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

Note that the $N_{i,2}$, restricted to the interval $u \in [0, 1]$, are the quadratic Bernstein polynomials (Section 1.3 and Figure 1.13b). For this reason, the B-spline representation with a knot vector of the form

$$U = \{\underbrace{0, \ldots, 0}_{p+1}, \underbrace{1, \ldots, 1}_{p+1}\}$$

is a generalization of the Bézier representation.

Ex2.2 Let $U = \{u_0 = 0, u_1 = 0, u_2 = 0, u_3 = 1, u_4 = 2, u_5 = 3, u_6 = 4, u_7 = 4, u_8 = 5, u_9 = 5, u_{10} = 5\}$ and $p = 2$. The zeroth-, first-, and second-degree basis functions are computed here. The ones not identically zero

are shown in Figures 2.4, 2.5, and 2.6, respectively

$$N_{0,0} = N_{1,0} = 0 \quad \text{for } -\infty < u < \infty$$

$$N_{2,0} = \begin{cases} 1 & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{3,0} = \begin{cases} 1 & 1 \le u < 2 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{4,0} = \begin{cases} 1 & 2 \le u < 3 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{5,0} = \begin{cases} 1 & 3 \le u < 4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{6,0} = 0 \quad \text{for } -\infty < u < \infty$$

$$N_{7,0} = \begin{cases} 1 & 4 \le u < 5 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{8,0} = N_{9,0} = 0 \quad \text{for } -\infty < u < \infty$$

$$N_{0,1} = \frac{u-0}{0-0}N_{0,0} + \frac{0-u}{0-0}N_{1,0} = 0 \qquad -\infty < u < \infty$$

$$N_{1,1} = \frac{u-0}{0-0}N_{1,0} + \frac{1-u}{1-0}N_{2,0} = \begin{cases} 1-u & 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{2,1} = \frac{u-0}{1-0}N_{2,0} + \frac{2-u}{2-1}N_{3,0} = \begin{cases} u & 0 \le u < 1 \\ 2-u & 1 \le u < 2 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{3,1} = \frac{u-1}{2-1}N_{3,0} + \frac{3-u}{3-2}N_{4,0} = \begin{cases} u-1 & 1 \le u < 2 \\ 3-u & 2 \le u < 3 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{4,1} = \frac{u-2}{3-2}N_{4,0} + \frac{4-u}{4-3}N_{5,0} = \begin{cases} u-2 & 2 \le u < 3 \\ 4-u & 3 \le u < 4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{5,1} = \frac{u-3}{4-3}N_{5,0} + \frac{4-u}{4-4}N_{6,0} = \begin{cases} u-3 & 3 \le u < 4 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{6,1} = \frac{u-4}{4-4}N_{6,0} + \frac{5-u}{5-4}N_{7,0} = \begin{cases} 5-u & 4 \le u < 5 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{7,1} = \frac{u-4}{5-4}N_{7,0} + \frac{5-u}{5-5}N_{8,0} = \begin{cases} u-4 & 4 \le u < 5 \\ 0 & \text{otherwise} \end{cases}$$

$$N_{8,1} = \frac{u-5}{5-5}N_{8,0} + \frac{5-u}{5-5}N_{9,0} = 0 \qquad -\infty < u < \infty$$

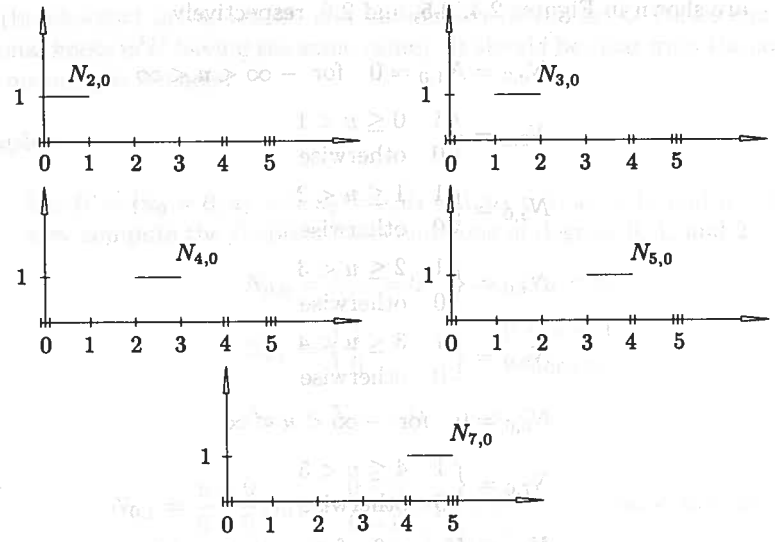


Figure 2.4. The nonzero zeroth-degree basis functions, $U = \{0,0,0,1,2,3,4,4,5,5,5\}$.

All of the following $N_{i,2}$ are zero everywhere except on the specified intervals, that is

$$N_{0,2} = \frac{u-0}{0-0}N_{0,1} + \frac{1-u}{1-0}N_{1,1} = (1-u)^2 \qquad 0 \le u < 1$$

$$N_{1,2} = \frac{u-0}{1-0}N_{1,1} + \frac{2-u}{2-0}N_{2,1} = \begin{cases} 2u - \frac{3}{2}u^2 & 0 \le u < 1 \\ \frac{1}{2}(2-u)^2 & 1 \le u < 2 \end{cases}$$

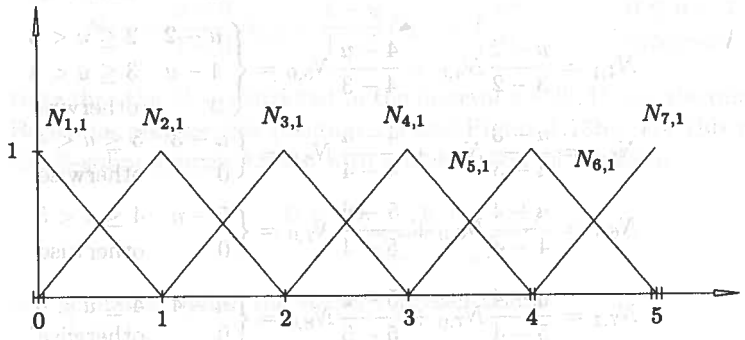


Figure 2.5. The nonzero first-degree basis functions, $U = \{0,0,0,1,2,3,4,4,5,5,5\}$.

$$N_{2,2} = \frac{u-0}{2-0}N_{2,1} + \frac{3-u}{3-1}N_{3,1} = \begin{cases} \frac{1}{2}u^2 & 0 \le u < 1 \\ -\frac{3}{2} + 3u - u^2 & 1 \le u < 2 \\ \frac{1}{2}(3-u)^2 & 2 \le u < 3 \end{cases}$$

$$N_{3,2} = \frac{u-1}{3-1}N_{3,1} + \frac{4-u}{4-2}N_{4,1} = \begin{cases} \frac{1}{2}(u-1)^2 & 1 \le u < 2 \\ -\frac{11}{2} + 5u - u^2 & 2 \le u < 3 \\ \frac{1}{2}(4-u)^2 & 3 \le u < 4 \end{cases}$$

$$N_{4,2} = \frac{u-2}{4-2}N_{4,1} + \frac{4-u}{4-3}N_{5,1} = \begin{cases} \frac{1}{2}(u-2)^2 & 2 \le u < 3 \\ -16 + 10u - \frac{3}{2}u^2 & 3 \le u < 4 \end{cases}$$

$$N_{5,2} = \frac{u-3}{4-3}N_{5,1} + \frac{5-u}{5-4}N_{6,1} = \begin{cases} (u-3)^2 & 3 \le u < 4 \\ (5-u)^2 & 4 \le u < 5 \end{cases}$$

$$N_{6,2} = \frac{u-4}{5-4}N_{6,1} + \frac{5-u}{5-4}N_{7,1} = 2(u-4)(5-u) \qquad 4 \le u < 5$$

$$N_{7,2} = \frac{u-4}{5-4}N_{7,1} + \frac{5-u}{5-5}N_{8,1} = (u-4)^2 \qquad 4 \le u < 5$$

We now list a number of important properties of the B-spline basis functions. As we see in the next chapter, it is these properties which determine the many desirable geometric characteristics in B-spline curves and surfaces. Assume degree p and a knot vector $U = \{u_0, \ldots, u_m\}$.

P2.1 $N_{i,p}(u) = 0$ if u is outside the interval $[u_i, u_{i+p+1})$ (local support property). This is illustrated by the triangular scheme shown here. Notice

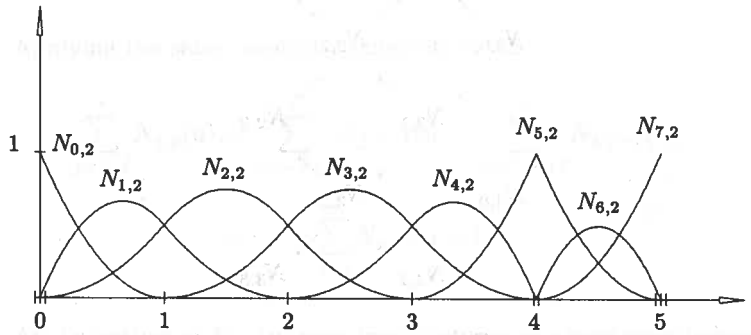
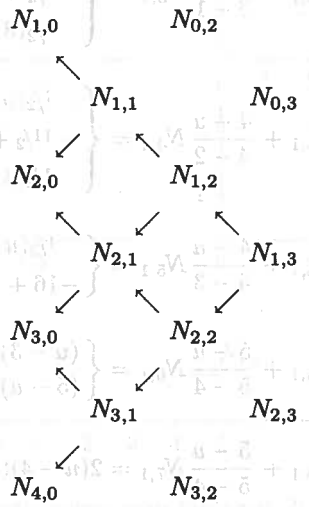
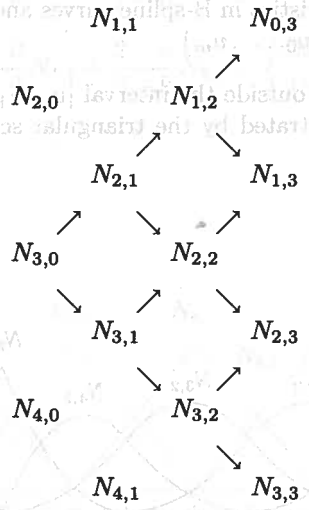


Figure 2.6. The nonzero second-degree basis functions, $U = \{0,0,0,1,2,3,4,4,5,5,5\}$.

that $N_{1,3}$ is a combination of $N_{1,0}$, $N_{2,0}$, $N_{3,0}$, and $N_{4,0}$. Thus, $N_{1,3}$ is nonzero only for $u \in [u_1, u_5)$



P2.2 In any given knot span, $[u_j, u_{j+1})$, at most $p+1$ of the $N_{i,p}$ are nonzero, namely the functions $N_{j-p,p}, \ldots, N_{j,p}$. On $[u_3, u_4)$ the only nonzero zeroth-degree function is $N_{3,0}$. Hence, the only cubic functions not zero on $[u_3, u_4)$ are $N_{0,3}, \ldots, N_{3,3}$. This property is illustrated here



P2.3 $N_{i,p}(u) \geq 0$ for all i, p, and u (nonnegativity). This is proven by induction on p. It is clearly true for $p = 0$; assume it is true for $p-1$, $p \geq 0$,

with i and u arbitrary. By definition

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \qquad (2.6)$$

By P2.1, $N_{i,p-1}(u) = 0$ if $u \notin [u_i, u_{i+p})$. But $u \in [u_i, u_{i+p})$ implies

$$\frac{u - u_i}{u_{i+p} - u_i}$$

is nonnegative. By assumption, $N_{i,p-1}(u)$ is nonnegative, and thus the first term of Eq. (2.6) is nonnegative. The same is true for the second term, and hence the $N_{i,p}(u)$ are nonnegative;

P2.4 For an arbitrary knot span, $[u_i, u_{i+1})$, $\sum_{j=i-p}^{i} N_{j,p}(u) = 1$ for all $u \in [u_i, u_{i+1})$ (partition of unity). To prove this, consider

$$\sum_{j=i-p}^{i} N_{j,p}(u) = \sum_{j=i-p}^{i} \frac{u - u_j}{u_{j+p} - u_j} N_{j,p-1}(u) + \sum_{j=i-p}^{i} \frac{u_{j+p+1} - u}{u_{j+p+1} - u_{j+1}} N_{j+1,p-1}(u)$$

Changing the summation variable in the second sum from $i-p$ to $i-p+1$, and considering that $N_{i-p,p-1}(u) = N_{i+1,p-1}(u) = 0$, we have

$$\sum_{j=i-p}^{i} N_{j,p}(u) = \sum_{j=i-p+1}^{i} \left[\frac{u - u_j}{u_{j+p} - u_j} + \frac{u_{j+p} - u}{u_{j+p} - u_j} \right] N_{j,p-1}(u)$$

$$= \sum_{j=i-p+1}^{i} N_{j,p-1}(u)$$

Applying the same concept recursively yields

$$\sum_{j=i-p}^{i} N_{j,p}(u) = \sum_{j=i-p+1}^{i} N_{j,p-1}(u) = \sum_{j=i-p+2}^{i} N_{j,p-2}(u)$$

$$= \cdots = \sum_{j=i}^{i} N_{j,0}(u) = 1$$

P2.5 All derivatives of $N_{i,p}(u)$ exist in the interior of a knot span (where it is a polynomial, see Figure 2.7). At a knot $N_{i,p}(u)$ is $p-k$ times continuously differentiable, where k is the multiplicity of the knot. Hence, increasing

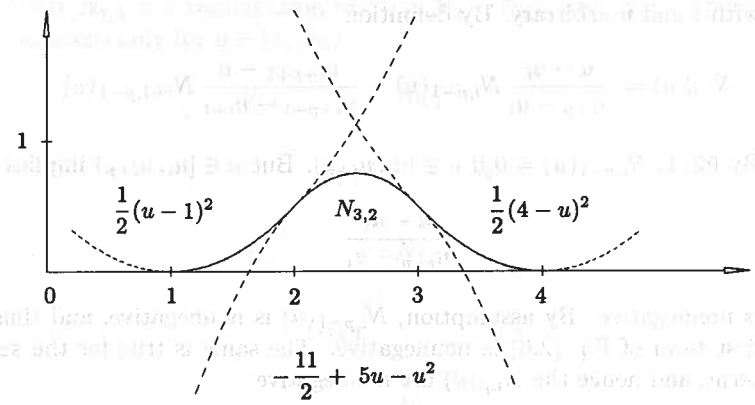


Figure 2.7. The decomposition of $N_{3,2}$ into its polynomial pieces (parabolas).

degree increases continuity, and increasing knot multiplicity decreases continuity;

P2.6 Except for the case $p = 0$, $N_{i,p}(u)$ attains exactly one maximum value.

It is important to understand the effect of multiple knots. Consider the functions $N_{0,2}$, $N_{1,2}$, $N_{2,2}$, $N_{5,2}$, and $N_{6,2}$ of Figure 2.6. Recalling that $U = \{0,0,0,1,2,3,4,4,5,5,5\}$, from Eq. (2.5) and P2.1, we see that these functions are computed on the following knot spans and are zero outside these spans

$$\begin{aligned} N_{0,2} &: \{0,0,0,1\} \\ N_{1,2} &: \{0,0,1,2\} \\ N_{2,2} &: \{0,1,2,3\} \\ N_{5,2} &: \{3,4,4,5\} \\ N_{6,2} &: \{4,4,5,5\} \end{aligned}$$

Now the word 'multiplicity' is understood in two different ways:

- the multiplicity of a knot in the knot vector;
- the multiplicity of a knot with respect to a specific basis function.

For example, $u = 0$ has multiplicity three in the previous knot vector U. But with respect to the functions $N_{0,2}$, $N_{1,2}$, $N_{2,2}$, and $N_{5,2}$, $u = 0$ is a knot of multiplicity 3, 2, 1, and 0, respectively. From P2.5, the continuity of these functions at $u = 0$ is $N_{0,2}$ discontinuous; $N_{1,2}$ C^0 continuous; $N_{2,2}$ C^1 continuous; and $N_{5,2}$ totally unaffected ($N_{5,2}$ and all its derivatives are zero at $u = 0$, from both sides). $N_{1,2}$ 'sees' $u = 0$ as a double knot, hence it is C^0 continuous. $N_{2,2}$ 'sees' all its knots with multiplicity 1, thus it is C^1 continuous everywhere. Clearly, another effect of multiple knots (as seen by the functions) is to reduce the number of 'apparent' intervals on which a function is nonzero; e.g., $N_{6,2}$ is nonzero only on $u \in [4,5)$, and it is only C^0 continuous at $u = 4$ and $u = 5$.

2.3 Derivatives of B-spline Basis Functions

The derivative of a basis function is given by

$$N'_{i,p} = \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.7}$$

(See Figure 2.8 for a graphical illustration.) We prove this by induction on p. For $p = 1$, $N_{i,p-1}$ and $N_{i+1,p-1}$ are either 0 or 1, and thus $N'_{i,p}$ is either

$$\frac{1}{u_{i+1} - u_i} \quad \text{or} \quad -\frac{1}{u_{i+2} - u_{i+1}}$$

(see Figure 2.5). Now assume that Eq. (2.7) is true for $p - 1$, $p > 1$. Using the product rule, $(fg)' = f'g + fg'$, to differentiate the basis function

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

yields

$$N'_{i,p} = \frac{1}{u_{i+p} - u_i} N_{i,p-1} + \frac{u - u_i}{u_{i+p} - u_i} N'_{i,p-1} - \frac{1}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1} + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N'_{i+1,p-1} \tag{2.8}$$

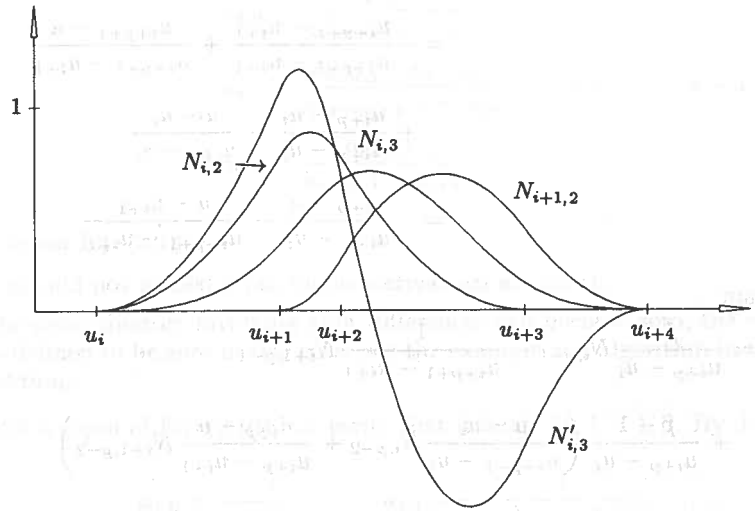


Figure 2.8. The recursive definition of B-spline derivatives.

Substituting Eq. (2.7) into Eq. (2.8) for $N'_{i,p-1}$ and $N'_{i+1,p-1}$ yields

$$\begin{aligned}
N'_{i,p} &= \frac{1}{u_{i+p}-u_i}N_{i,p-1} - \frac{1}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1} \\
&\quad + \frac{u-u_i}{u_{i+p}-u_i}\left(\frac{p-1}{u_{i+p-1}-u_i}N_{i,p-2} - \frac{p-1}{u_{i+p}-u_{i+1}}N_{i+1,p-2}\right) \\
&\quad + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}}\left(\frac{p-1}{u_{i+p}-u_{i+1}}N_{i+1,p-2} - \frac{p-1}{u_{i+p+1}-u_{i+2}}N_{i+2,p-2}\right) \\
&= \frac{1}{u_{i+p}-u_i}N_{i,p-1} - \frac{1}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1} \\
&\quad + \frac{p-1}{u_{i+p}-u_i}\frac{u-u_i}{u_{i+p-1}-u_i}N_{i,p-2} \\
&\quad + \frac{p-1}{u_{i+p}-u_{i+1}}\left(\frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} - \frac{u-u_i}{u_{i+p}-u_i}\right)N_{i+1,p-2} \\
&\quad - \frac{p-1}{u_{i+p+1}-u_{i+1}}\frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+2}}N_{i+2,p-2}
\end{aligned}$$

Noting that

$$\begin{aligned}
\frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} - \frac{u-u_i}{u_{i+p}-u_i} &= -1 + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} + 1 - \frac{u-u_i}{u_{i+p}-u_i} \\
&= -\frac{u_{i+p+1}-u_{i+1}}{u_{i+p+1}-u_{i+1}} + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} \\
&\quad + \frac{u_{i+p}-u_i}{u_{i+p}-u_i} - \frac{u-u_i}{u_{i+p}-u_i} \\
&= \frac{u_{i+p}-u}{u_{i+p}-u_i} - \frac{u-u_{i+1}}{u_{i+p+1}-u_{i+1}}
\end{aligned}$$

we obtain

$$\begin{aligned}
N'_{i,p} &= \frac{1}{u_{i+p}-u_i}N_{i,p-1} - \frac{1}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1} \\
&\quad + \frac{p-1}{u_{i+p}-u_i}\left(\frac{u-u_i}{u_{i+p-1}-u_i}N_{i,p-2} + \frac{u_{i+p}-u}{u_{i+p}-u_{i+1}}N_{i+1,p-2}\right) \\
&\quad - \frac{p-1}{u_{i+p+1}-u_{i+1}}\left(\frac{u-u_{i+1}}{u_{i+p}-u_{i+1}}N_{i+1,p-2} + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+2}}N_{i+2,p-2}\right)
\end{aligned}$$

By the Cox-deBoor formula (Eq. [2.5]), the expressions in the parentheses can be replaced by $N_{i,p-1}$ and $N_{i+1,p-1}$, respectively. It follows that

$$\begin{aligned}
N'_{i,p} &= \frac{1}{u_{i+p}-u_i}N_{i,p-1} - \frac{1}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1} \\
&\quad + \frac{p-1}{u_{i+p}-u_i}N_{i,p-1} - \frac{p-1}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1} \\
&= \frac{p}{u_{i+p}-u_i}N_{i,p-1} - \frac{p}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}
\end{aligned}$$

This completes the proof.

Now let $N^{(k)}_{i,p}$ denote the kth derivative of $N_{i,p}(u)$. Repeated differentiation of Eq. (2.7) produces the general formula

$$N^{(k)}_{i,p}(u) = p\left(\frac{N^{(k-1)}_{i,p-1}}{u_{i+p}-u_i} - \frac{N^{(k-1)}_{i+1,p-1}}{u_{i+p+1}-u_{i+1}}\right) \tag{2.9}$$

Equation (2.10) is another generalization of Eq. (2.7). It computes the kth derivative of $N_{i,p}(u)$ in terms of the functions $N_{i,p-k}, \ldots, N_{i+k,p-k}$

$$N^{(k)}_{i,p} = \frac{p!}{(p-k)!}\sum_{j=0}^{k} a_{k,j}N_{i+j,p-k} \tag{2.10}$$

with

$$\begin{aligned}
a_{0,0} &= 1 \\
a_{k,0} &= \frac{a_{k-1,0}}{u_{i+p-k+1}-u_i} \\
a_{k,j} &= \frac{a_{k-1,j}-a_{k-1,j-1}}{u_{i+p+j-k+1}-u_{i+j}} \qquad j = 1, \ldots, k-1 \\
a_{k,k} &= \frac{-a_{k-1,k-1}}{u_{i+p+1}-u_{i+k}}
\end{aligned}$$

Remarks on Eq. (2.10):

- k should not exceed p (all higher derivatives are zero);
- the denominators involving knot differences can become zero; the quotient is defined to be zero in this case (see the example and algorithm in the next section).

We omit a proof of Eq. (2.10) but verify that it holds for $k = 1, 2$. By definition

$$a_{1,0} = \frac{1}{u_{i+p}-u_i} \qquad a_{1,1} = -\frac{1}{u_{i+p+1}-u_{i+1}}$$

and

$$N^{(1)}_{i,p} = 2(a_{1,0}N_{i,p-1} + a_{1,1}N_{i+1,p-1})$$

Comparing this with Eq. (2.7) proves the case for $k = 1$; now let $k = 2$. Differentiating Eq. (2.7) yields

$$N_{i,p}^{(2)} = \frac{p}{u_{i+p} - u_i} N_{i,p-1}^{(1)} - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}^{(1)}$$

$$= \frac{p}{u_{i+p} - u_i} \left(\frac{p-1}{u_{i+p-1} - u_i} N_{i,p-2} - \frac{p-1}{u_{i+p} - u_{i+1}} N_{i+1,p-2} \right)$$

$$- \frac{p}{u_{i+p+1} - u_{i+1}} \left(\frac{p-1}{u_{i+p} - u_{i+1}} N_{i+1,p-2} - \frac{p-1}{u_{i+p+1} - u_{i+2}} N_{i+2,p-2} \right)$$

$$= p(p-1) \left[\frac{a_{1,0}}{u_{i+p-1} - u_i} N_{i,p-2} \right.$$

$$- \frac{1}{u_{i+p} - u_{i+1}} \left(\frac{1}{u_{i+p} - u_i} + \frac{1}{u_{i+p+1} - u_{i+1}} \right) N_{i+1,p-2}$$

$$\left. + \frac{a_{1,1}}{u_{i+p+1} - u_{i+2}} N_{i+2,p-2} \right]$$

$$= p(p-1) \left(a_{2,0} N_{i,p-2} + \frac{a_{1,1} - a_{1,0}}{u_{i+p} - u_{i+1}} N_{i+1,p-2} + a_{2,2} N_{i+2,p-2} \right)$$

Noting that $k = 2$ and

$$a_{2,1} = \frac{a_{1,1} - a_{1,0}}{u_{i+p} - u_{i+1}}$$

it follows that

$$N_{i,p}^{(2)}(u) = 2 \sum_{j=0}^{2} a_{2,j} N_{i+j,p-2}(u)$$

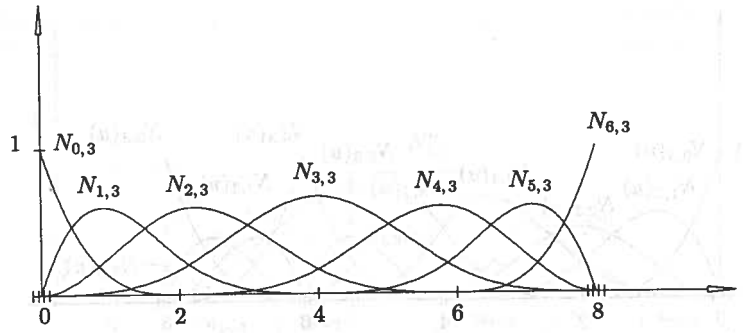
For completeness, we give an additional formula for computing derivatives of the B-spline basis functions (see [Butt76])

$$N_{i,p}^{(k)} = \frac{p}{p-k} \left(\frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}^{(k)} + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}^{(k)} \right)$$

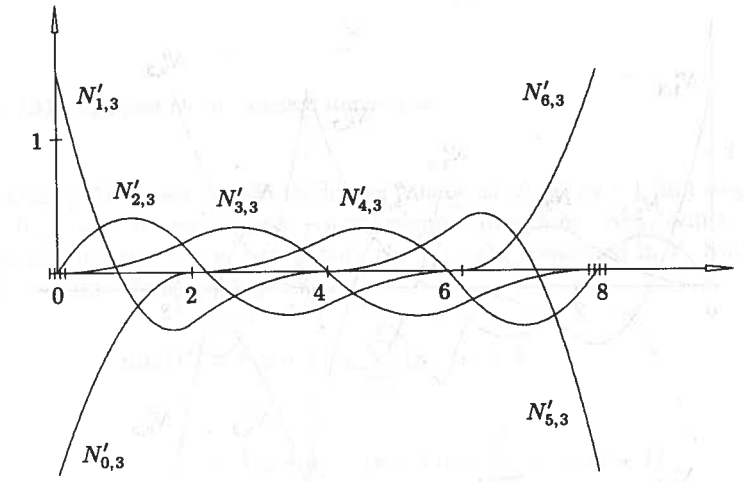
$$k = 0, \ldots, p-1 \qquad (2.11)$$

Equation (2.11) gives the kth derivative of $N_{i,p}(u)$ in terms of the kth derivative of $N_{i,p-1}$ and $N_{i+1,p-1}$.

Figures 2.9b and 2.10b show the derivatives corresponding to the basis functions in Figures 2.9a and 2.10a. Figure 2.11 shows all the nonzero derivatives of $N_{i,3}$. Note the effect of multiple knots in Figure 2.10b; $N'_{6,3}$ has a jump at the triple knot.



(a)



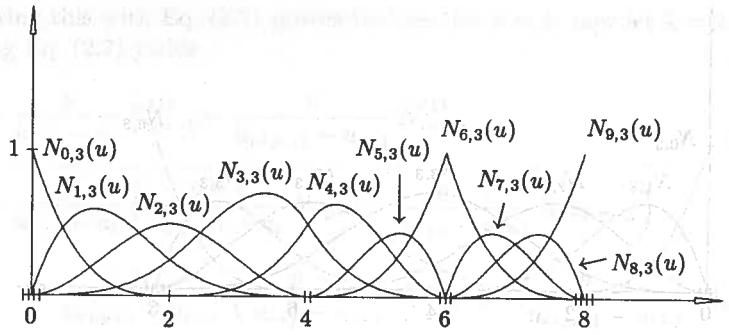
(b)

Figure 2.9. (a) Cubic basis functions; (b) derivatives corresponding to the basis functions in Figure 2.9a.

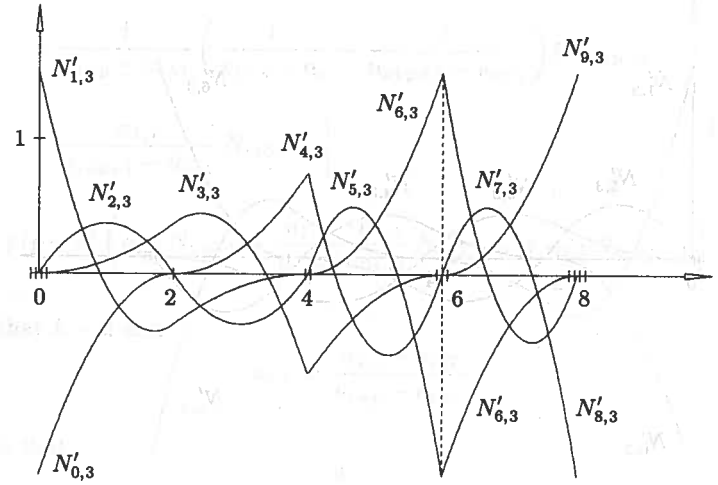
2.4 Further Properties of the Basis Functions

Let $\{u_j\}$, $0 \le j \le k$, be a strictly increasing set of breakpoints. The set of all piecewise polynomial functions of degree p on $\{u_j\}$ which are C^{r_j} continuous at $u = u_j$ forms a vector space, $\mathcal{V}$ ($-1 \le r_j \le p$). If no continuity constraints are imposed ($r_j = -1$ for all j), then the dimension of $\mathcal{V}$ (denoted $\dim(\mathcal{V})$) is equal to $k(p+1)$. Each continuity constraint decreases the dimension by one, thus

$$\dim(\mathcal{V}) = k(p+1) - \sum_{j=0}^{k} (r_j + 1) \qquad (2.12)$$



(a)



(b)

Figure 2.10. (a) Cubic basis functions showing single, double, and triple knots; (b) derivatives of the functions in Figure 2.10a.

By Property P2.5, we obtain the B-spline basis functions of p-degree with knots at the $\{u_j\}$, and with the desired continuity, by setting the appropriate knot multiplicities, s_j, where $s_j = p - r_j$. Hence, we use a knot vector of the form

$$U = \{\underbrace{u_0, \ldots, u_0}_{s_0}, \underbrace{u_1, \ldots, u_1}_{s_1}, \ldots, \underbrace{u_k, \ldots, u_k}_{s_k}\}$$

Now set

$$m = \left(\sum_{j=0}^{k} s_j\right) - 1$$

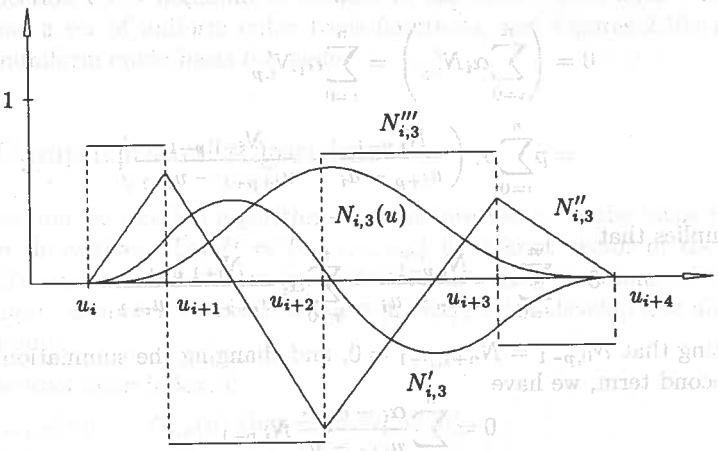


Figure 2.11. $N_{i,3}$ and all its nonzero derivatives.

Then clearly, there are m zeroth-degree functions, $N_{i,0}$, $m-1$ first degree functions, $N_{i,1}$, and in general, $m-p$ pth-degree functions, $N_{i,p}$, which have the desired continuity, $r_j = p - s_j$. Hence the $N_{i,p}$ are contained in $\mathcal{V}$. Substituting $s_j = p - r_j$ into Eq. (2.12) yields

$$\begin{aligned}\dim(\mathcal{V}) &= k(p+1) - \sum_{j=0}^{k}(p - s_j + 1) \\ &= k(p+1) - (k+1)p + \sum_{j=0}^{k} s_j - (k+1) \\ &= -p - 1 + \sum_{j=0}^{k} s_j \\ &= m - p\end{aligned}$$

Thus, the number of pth-degree B-spline basis functions on U is equal to $\dim(\mathcal{V})$. We now justify the term 'basis' functions by showing that the $N_{i,p}$ are linearly independent, i.e., they form a basis for the vector space, $\mathcal{V}$. The proof is by induction on p. Clearly, the zeroth-degree functions are linearly independent. Assume the $(p-1)$th-degree functions are linearly independent for $p > 0$. Set $n = m - p - 1$, and assume that

$$\sum_{i=0}^{n} \alpha_i N_{i,p}(u) = 0 \quad \text{for all } u$$

Using Eq. (2.7) we obtain

$$0 = \left(\sum_{i=0}^{n} \alpha_i N'_{i,p}\right)' = \sum_{i=0}^{n} \alpha_i N'_{i,p}$$

$$= p\sum_{i=0}^{n} \alpha_i \left(\frac{N_{i,p-1}}{u_{i+p} - u_i} - \frac{N_{i+1,p-1}}{u_{i+p+1} - u_{i+1}}\right)$$

which implies that

$$0 = \sum_{i=0}^{n} \alpha_i \frac{N_{i,p-1}}{u_{i+p} - u_i} - \sum_{i=0}^{n} \alpha_i \frac{N_{i+1,p-1}}{u_{i+p+1} - u_{i+1}}$$

Now noting that $N_{0,p-1} = N_{n+1,p-1} = 0$, and changing the summation variable in the second term, we have

$$0 = \sum_{i=1}^{n} \frac{\alpha_i - \alpha_{i-1}}{u_{i+p} - u_i} N_{i,p-1}$$

which implies $\alpha_i - \alpha_{i-1} = 0$ for all i (by assumption), which in turn implies $\alpha_i = 0$ for all i. This completes the proof.

We turn our attention now to knot vectors. Clearly, once the degree is fixed the knot vector completely determines the functions $N_{i,p}(u)$. There are several types of knot vectors, and unfortunately terminology varies in the literature. In this book we consider only *nonperiodic* (or *clamped* or *open*) knot vectors, which have the form

$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\} \tag{2.13}$$

that is, the first and last knots have multiplicity $p + 1$. For nonperiodic knot vectors we have two additional properties of the basis functions:

P2.7 A knot vector of the form

$$U = \{\underbrace{0, \ldots, 0}_{p+1}, \underbrace{1, \ldots, 1}_{p+1}\}$$

yields the Bernstein polynomials of degree p (see Example Ex2.1 in Section 2.2);

P2.8 Let $m+1$ be the number of knots. Then there are $n+1$ basis functions, where $n = m - p - 1$; $N_{0,p}(a) = 1$ and $N_{n,p}(b) = 1$. For example, $N_{0,p}(a) = 1$ follows from the fact that $N_{0,0}, \ldots, N_{p-1,0} = 0$, since this implies that $N_{0,p}(a) = N_{p,0}(a) = 1$. From P2.4 it follows that $N_{i,p}(a) = 0$ for $i \neq 0$, and $N_{i,p}(b) = 0$ for $i \neq n$.

For the remainder of this book, all knot vectors are understood to be nonperiodic. We define a knot vector $U = \{u_0, \ldots, u_m\}$ to be *uniform* if all interior knots are equally spaced, i.e., if there exists a real number, d, such that $d = u_{i+1} - u_i$ for all $p \leq i \leq m-p-1$; otherwise it is *nonuniform*. The knot vector of Example

Ex2.2, Section 2.2 is nonuniform because of the double knot at $u = 4$. Figure 2.9a shows a set of uniform cubic basis functions, and Figures 2.10a and 2.12 show nonuniform cubic basis functions.

2.5 Computational Algorithms

In this section we develop algorithms to compute values of the basis functions and their derivatives. Let $U = \{u_0, \ldots, u_m\}$ be a knot vector of the form in Eq. (2.13), and assume we are interested in the basis functions of degree p. Furthermore, assume u is fixed, and $u \in [u_i, u_{i+1})$. We develop five algorithms that compute:

- the knot span index, i;
- $N_{i-p,p}(u), \ldots, N_{i,p}(u)$ (based on Eq. [2.5]);
- $N^{(k)}_{i-p,p}(u), \ldots, N^{(k)}_{i,p}(u)$ for $k = 0, \ldots, p$; for $k > p$ the derivatives are zero (this algorithm is based on Eq. [2.10]);
- a single basis function, $N_{j,p}(u)$, where $0 \leq j \leq m - p - 1$;
- the derivatives of a single basis function, $N^{(k)}_{j,p}(u)$, where $0 \leq j \leq m-p-1$ and $k = 0, \ldots, p$ (based on Eq. [2.9]).

We present the two algorithms which compute $p + 1$ functions before the two which compute only one, because they are the most important and actually are somewhat simpler.

From P2.2 and the assumption that $u \in [u_i, u_{i+1})$, it follows that we can focus our attention on the functions $N_{i-p,p}, \ldots, N_{i,p}$ and their derivatives; all other functions are identically zero, and it is wasteful to actually compute them. Hence, the first step in evaluation is to determine the knot span in which u lies. Either a linear or a binary search of the knot vector can be used; we present here a binary

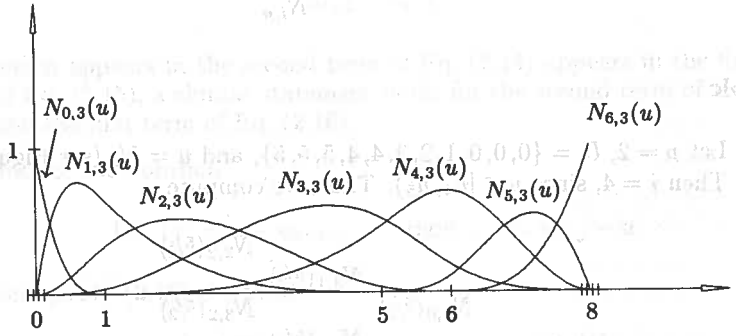


Figure 2.12. Nonuniform cubic basis functions defined on $U = \{0, 0, 0, 0, 1, 5, 6, 8, 8, 8, 8\}$.

search. Since we are using intervals of the form $u \in [u_i, u_{i+1})$, a subtle problem in the evaluation of the basis functions is the special case $u = u_m$. It is best to handle this at the lowest level by setting the span index to n $(= m - p - 1)$. Hence, in this case $u \in (u_{m-p-1}, u_{m-p}]$. `FindSpan` is an integer function which returns the span index.

```
ALGORITHM A2.1
  int  FindSpan(n,p,u,U)
  { /*  Determine the knot span index  */
    /*  Input:  n,p,u,U  */
    /*  Return: the knot span index  */
    if (u == U[n+1])  return(n);   /* Special case */
    low = p;   high = n+1;   /* Do binary search */
    mid =(low+high)/2;
    while (u < U[mid] || u >= U[mid+1])
      {
      if (u < U[mid])   high = mid;
       else             low = mid;
      mid = (low+high)/2;
      }
    return(mid);
    }
```

Now we tackle the second algorithm. Assuming u is in the ith span, computation of the nonzero functions results in an inverted triangular scheme

$$
\begin{array}{ccccc}
 & & & & N_{i-p,p} \\
 & & & & \\
 & & & & N_{i-1,1} \\
N_{i,0} & & \cdots & & \vdots \\
 & & & & N_{i,1} \\
 & & & & \\
 & & & & N_{i,p}
\end{array}
$$

Example

Ex2.3 Let $p = 2$, $U = \{0,0,0,1,2,3,4,4,5,5,5\}$, and $u = 5/2$ (see Figure 2.6). Then $i = 4$, since $u \in [u_4, u_5)$. Thus, we compute

$$
\begin{array}{ccccc}
 & & & & N_{2,2}(5/2) \\
 & & & N_{3,1}(5/2) & \\
N_{4,0}(5/2) & & & & N_{3,2}(5/2) \\
 & & & N_{4,1}(5/2) & \\
 & & & & N_{4,2}(5/2)
\end{array}
$$

Substituting $u = 5/2$ into Eq. (2.5) (the reader should do this) yields

$$N_{4,0}\left(\frac{5}{2}\right) = 1$$

$$N_{3,1}\left(\frac{5}{2}\right) = \frac{1}{2} \qquad N_{4,1}\left(\frac{5}{2}\right) = \frac{1}{2}$$

$$N_{2,2}\left(\frac{5}{2}\right) = \frac{1}{8} \qquad N_{3,2}\left(\frac{5}{2}\right) = \frac{6}{8} \qquad N_{4,2}\left(\frac{5}{2}\right) = \frac{1}{8}$$

Notice that for fixed degree the functions sum to 1 (P2.4).

It will be clear to the reader who carried out the substitutions in this example that there is a great deal of redundant computation inherent in Eq. (2.5). For example, writing out the second-degree functions in general terms, we have

$$N_{i-2,2}(u) = \frac{u - u_{i-2}}{u_i - u_{i-2}} N_{i-2,1}(u) + \frac{u_{i+1} - u}{u_{i+1} - u_{i-1}} N_{i-1,1}(u) \tag{2.14}$$

$$N_{i-1,2}(u) = \frac{u - u_{i-1}}{u_{i+1} - u_{i-1}} N_{i-1,1}(u) + \frac{u_{i+2} - u}{u_{i+2} - u_i} N_{i,1}(u) \tag{2.15}$$

$$N_{i,2}(u) = \frac{u - u_i}{u_{i+2} - u_i} N_{i,1}(u) + \frac{u_{i+3} - u}{u_{i+3} - u_{i+1}} N_{i+1,1}(u) \tag{2.16}$$

Note that

- the first term of Eq. (2.14) and the last term of Eq. (2.16) are not computed, since $N_{i-2,1}(u) = N_{i+1,1}(u) = 0$;
- the expression

$$\frac{N_{i-1,1}(u)}{u_{i+1} - u_{i-1}}$$

  which appears in the second term of Eq. (2.14) appears in the first term of Eq. (2.15); a similar statement holds for the second term of Eq. (2.15) and the first term of Eq. (2.16).

We introduce the notation

$$\text{left}\,[j] = u - u_{i+1-j} \qquad \text{right}\,[j] = u_{i+j} - u$$

Equations (2.14)–(2.16) are then

$$N_{i-2,2}(u) = \frac{\text{left}\,[3]}{\text{right}\,[0] + \text{left}\,[3]} N_{i-2,1}(u) + \frac{\text{right}\,[1]}{\text{right}\,[1] + \text{left}\,[2]} N_{i-1,1}(u)$$

$$N_{i-1,2}(u) = \frac{\text{left [2]}}{\text{right [1]} + \text{left [2]}} N_{i-1,1}(u) + \frac{\text{right [2]}}{\text{right [2]} + \text{left [1]}} N_{i,1}(u)$$

$$N_{i,2}(u) = \frac{\text{left [1]}}{\text{right [2]} + \text{left [1]}} N_{i,1}(u) + \frac{\text{right [3]}}{\text{right [3]} + \text{left [0]}} N_{i+1,1}(u)$$

Based on these observations, Algorithm A2.2 computes all the nonvanishing basis functions and stores them in the array N[0],...,N[p].

```
ALGORITHM A2.2
  BasisFuns(i,u,p,U,N)
    { /*  Compute the nonvanishing basis functions  */
      /*  Input:  i,u,p,U  */
      /*  Output: N  */
    N[0]=1.0;
    for (j=1; j<=p; j++)
      {
      left[j] = u-U[i+1-j];
      right[j] = U[i+j]-u;
      saved = 0.0;
      for (r=0; r<j; r++)
        {
        temp = N[r]/(right[r+1]+left[j-r]);
        N[r] = saved+right[r+1]*temp;
        saved = left[j-r]*temp;
        }
      N[j] = saved;
      }
    }
```

We remark that Algorithm A2.2 is not only efficient, but it also guarantees that there will be no division by zero, which can occur with a direct application of Eq. (2.5).

Now to the third algorithm; in particular, we want to compute all $N_{r,p}^{(k)}(u)$, for $i-p \le r \le i$ and $0 \le k \le n$, where $n \le p$. Inspection of Eq. (2.10) reveals that the basic ingredients are:

- the inverted triangle of nonzero basis functions computed in Algorithm A2.2;
- differences of knots (the sums: right[r+1]+left[j-r]), also computed in Algorithm A2.2;
- differences of the $a_{k,j}$; note that the $a_{k,j}$ depend on the $a_{k-1,j}$ but not the $a_{s,j}$, for $s < k-1$.

Viewed as a two-dimensional array of dimension $(p+1) \times (p+1)$, the basis functions fit into the upper triangle (including the diagonal), and the knot differences fit into the lower triangle, that is

$N_{i,0}(u)$	$N_{i-1,1}(u)$	$N_{i-2,2}(u)$
$u_{i+1} - u_i$	$N_{i,1}(u)$	$N_{i-1,2}(u)$
$u_{i+1} - u_{i-1}$	$u_{i+2} - u_i$	$N_{i,2}(u)$

Example

Ex2.4 Let $p = 2$, $U = \{0,0,0,1,2,3,4,4,5,5,5\}$, and $u = 5/2$. Then $u \in [u_4, u_5)$, and the array becomes

$N_{4,0}\left(\frac{5}{2}\right) = 1$	$N_{3,1}\left(\frac{5}{2}\right) = \frac{1}{2}$	$N_{2,2}\left(\frac{5}{2}\right) = \frac{1}{8}$
$u_5 - u_4 = 1$	$N_{4,1}\left(\frac{5}{2}\right) = \frac{1}{2}$	$N_{3,2}\left(\frac{5}{2}\right) = \frac{6}{8}$
$u_5 - u_3 = 2$	$u_6 - u_4 = 2$	$N_{4,2}\left(\frac{5}{2}\right) = \frac{1}{8}$

Now compute $N_{4,2}^{(1)}(5/2)$ and $N_{4,2}^{(2)}(5/2)$; with $i = 4$ in Eq. (2.10), we have

$$a_{1,0} = \frac{1}{u_6 - u_4} = \frac{1}{2}$$

$$a_{1,1} = -\frac{1}{u_7 - u_5} = -1$$

$$a_{2,0} = \frac{a_{1,0}}{u_5 - u_4} = \frac{1}{2}$$

$$a_{2,1} = \frac{a_{1,1} - a_{1,0}}{u_6 - u_5} = \frac{-1 - \frac{1}{2}}{4-3} = -\frac{3}{2}$$

$$a_{2,2} = -\frac{a_{1,1}}{u_7 - u_6} = \frac{1}{4-4} = \frac{1}{0}$$

$$N_{4,2}^{(1)} = 2\left[a_{1,0}N_{4,1}\left(\frac{5}{2}\right) + a_{1,1}N_{5,1}\left(\frac{5}{2}\right)\right]$$

and

$$N_{4,2}^{(2)} = 2\left[a_{2,0}N_{4,0}\left(\frac{5}{2}\right) + a_{2,1}N_{5,0}\left(\frac{5}{2}\right) + a_{2,2}N_{6,0}\left(\frac{5}{2}\right)\right]$$

Now $a_{1,1}$, $a_{2,1}$, and $a_{2,2}$ all use knot differences which are not in the array, but they are multiplied respectively by $N_{5,1}(5/2)$, $N_{5,0}(5/2)$, and

$N_{6,0}(5/2)$, which are also not in the array. These terms are defined to be zero, and we are left with

$$N_{4,2}^{(1)} = 2a_{1,0}N_{4,1}\left(\frac{5}{2}\right) = \frac{1}{2}$$

$$N_{4,2}^{(2)} = 2a_{2,0}N_{4,0}\left(\frac{5}{2}\right) = 1$$

To check these values, recall from Section 2.2 that $N_{4,2}(u) = \frac{1}{2}(u-2)^2$ on $u \in [2,3)$. The computation of $N_{3,2}^{(1)}(5/2)$, $N_{3,2}^{(2)}(5/2)$, $N_{2,2}^{(1)}(5/2)$, and $N_{2,2}^{(2)}(5/2)$ is analogous.

Based on these observations (and Ex2.4), it is not difficult to develop Algorithm A2.3, which computes the nonzero basis functions and their derivatives, up to and including the nth derivative ($n \le p$). Output is in the two-dimensional array, `ders`. `ders[k][j]` is the kth derivative of the function $N_{i-p+j,p}$, where $0 \le k \le n$ and $0 \le j \le p$. Two local arrays are used:

- `ndu[p+1][p+1]`, to store the basis functions and knot differences;
- `a[2][p+1]`, to store (in an alternating fashion) the two most recently computed rows $a_{k,j}$ and $a_{k-1,j}$.

The algorithm avoids division by zero and/or the use of terms not in the array `ndu[][]`.

```
ALGORITHM A2.3
  DersBasisFuns(i,u,p,n,U,ders)
    { /*  Compute nonzero basis functions and their  */
      /*  derivatives.  First section is A2.2 modified  */
      /*  to store functions and knot differences.  */
      /*  Input:  i,u,p,n,U  */
      /*  Output: ders  */
    ndu[0][0]=1.0;
    for (j=1; j<=p; j++)
      {
      left[j] = u-U[i+1-j];
      right[j] = U[i+j]-u;
      saved = 0.0;
      for (r=0; r<j; r++)
        {                                   /* Lower triangle */
        ndu[j][r] = right[r+1]+left[j-r];
        temp = ndu[r][j-1]/ndu[j][r];
                                            /* Upper triangle */
        ndu[r][j] = saved+right[r+1]*temp;
        saved = left[j-r]*temp;
        }
      ndu[j][j] = saved;
```



```
      }
    for (j=0; j<=p; j++)    /* Load the basis functions */
      ders[0][j] = ndu[j][p];
    /* This section computes the derivatives (Eq. [2.9]) */
    for (r=0; r<=p; r++)    /* Loop over function index */
      {
      s1=0;   s2=1;    /* Alternate rows in array a */
      a[0][0] = 1.0;
      /* Loop to compute kth derivative */
      for (k=1; k<=n; k++)
        {
        d = 0.0;
        rk = r-k;     pk = p-k;
        if (r >= k)
          {
          a[s2][0] = a[s1][0]/ndu[pk+1][rk];
          d = a[s2][0]*ndu[rk][pk];
          }
        if (rk >= -1)    j1 = 1;
          else           j1 = -rk;
        if (r-1 <= pk)   j2 = k-1;
          else           j2 = p-r;
        for (j=j1; j<=j2; j++)
          {
          a[s2][j] = (a[s1][j]-a[s1][j-1])/ndu[pk+1][rk+j];
          d += a[s2][j]*ndu[rk+j][pk];
          }
        if (r <= pk)
          {
          a[s2][k] = -a[s1][k-1]/ndu[pk+1][r];
          d += a[s2][k]*ndu[r][pk];
          }
        ders[k][r] = d;
        j=s1;   s1=s2;   s2=j;    /* Switch rows */
        }
      }
    /* Multiply through by the correct factors */
    /* (Eq. [2.9]) */
    r = p;
    for (k=1; k<=n; k++)
      {
      for (j=0; j<=p; j++)   ders[k][j] *= r;
      r *= (p-k);
      }
    }
```

We turn our attention now to the last two algorithms, namely computing a single basis function, $N_{i,p}(u)$, or the derivatives, $N_{i,p}^{(k)}(u)$, of a single basis function. The solutions to these problems result in triangular tables of the form

$$\begin{array}{ccccc} N_{i,0} & & & & \\ & N_{i,1} & & & \\ N_{i+1,0} & & N_{i,2} & & \\ \vdots & & \cdots & & N_{i,p} \\ N_{i+p-1,0} & & N_{i+p-2,2} & & \\ & N_{i+p-1,1} & & & \\ N_{i+p,0} & & & & \end{array}$$

Example

Ex2.5 Let $p = 2$, $U = \{0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5\}$, and $u = 5/2$. The computation of $N_{3,2}(5/2)$ yields

$$\begin{array}{ccc} N_{3,0}(5/2) = 0 & & \\ & N_{3,1}(5/2) = \frac{1}{2} & \\ N_{4,0}(5/2) = 1 & & N_{3,2}(5/2) = \frac{6}{8} \\ & N_{4,1}(5/2) = \frac{1}{2} & \\ N_{5,0}(5/2) = 0 & & \end{array}$$

$N_{4,2}(5/2)$ is obtained from

$$\begin{array}{ccc} N_{4,0}(5/2) = 1 & & \\ & N_{4,1}(5/2) = \frac{1}{2} & \\ N_{5,0}(5/2) = 0 & & N_{4,2}(5/2) = \frac{1}{8} \\ & N_{5,1}(5/2) = 0 & \\ N_{6,0}(5/2) = 0 & & \end{array}$$

Notice that the position and relative number of nonzero entries in the table depend on p and on the position of the 1 in the first column. Algorithm `A2.4` computes only the nonzero entries. The value $N_{i,p}(u)$ is returned in `Nip`; `m` is the high index of `U` ($m + 1$ knots). The algorithm is similar to Algorithm `A2.2` in its use of the variables `temp` and `saved`.

```
ALGORITHM A2.4
  OneBasisFun(p,m,U,i,u,Nip)
    { /*  Compute the basis function Nip  */
      /*  Input:  p,m,U,i,u  */
      /*  Output: Nip  */
```



```
if ((i == 0 && u == U[0]) ||      /* Special */
    (i == m-p-1 && u == U[m]))    /*  cases  */
  {
  Nip = 1.0;     return;
  }
if (u < U[i] || u >= U[i+p+1])    /* Local property */
  {
  Nip = 0.0;     return;
  }
for (j=0; j<=p; j++) /* Initialize zeroth-degree functs */
  if (u >= U[i+j] && u < U[i+j+1])  N[j] = 1.0;
    else            N[j] = 0.0;
for (k=1; k<=p; k++)  /* Compute triangular table */
  {
  if (N[0] == 0.0)  saved = 0.0;
    else        saved = ((u-U[i])*N[0])/(U[i+k]-U[i]);
  for (j=0; j<p-k+1; j++)
    {
    Uleft = U[i+j+1];
    Uright = U[i+j+k+1];
    if (N[j+1] == 0.0)
      {
      N[j] = saved;    saved = 0.0;
      }
      else
      {
      temp = N[j+1]/(Uright-Uleft);
      N[j] = saved+(Uright-u)*temp;
      saved = (u-Uleft)*temp;
      }
    }
  }
Nip = N[0];
}
```

Now for fixed i, the computation of the derivatives, $N_{i,p}^{(k)}(u)$, for $k = 0, \ldots, n$, $n \le p$, uses Eq. (2.9). For example, if $p = 3$ and $n = 3$, then

$$N_{i,3}^{(1)} = 3\left(\frac{N_{i,2}}{u_{i+3} - u_i} - \frac{N_{i+1,2}}{u_{i+4} - u_{i+1}}\right)$$

$$N_{i,3}^{(2)} = 3\left(\frac{N_{i,2}^{(1)}}{u_{i+3} - u_i} - \frac{N_{i+1,2}^{(1)}}{u_{i+4} - u_{i+1}}\right)$$

$$N_{i,3}^{(3)} = 3\left(\frac{N_{i,2}^{(2)}}{u_{i+3} - u_i} - \frac{N_{i+1,2}^{(2)}}{u_{i+4} - u_{i+1}}\right)$$

Using triangular tables, we must compute

$k = 0$:

$$\begin{array}{ccccc} N_{i,0} & & & & \\ & N_{i,1} & & & \\ N_{i+1,0} & & N_{i,2} & & \\ & N_{i+1,1} & & N_{i,3} & \\ N_{i+2,0} & & N_{i+1,2} & & \\ & N_{i+2,1} & & & \\ N_{i+3,0} & & & & \end{array}$$

$k = 1$:

$$\begin{array}{ccc} & N_{i,2} & \\ & & N_{i,3}^{(1)} \\ & N_{i+1,2} & \end{array}$$

$k = 2$:

$$\begin{array}{ccc} N_{i,1} & & \\ & N_{i,2}^{(1)} & \\ N_{i+1,1} & & N_{i,3}^{(2)} \\ & N_{i+1,2}^{(1)} & \\ N_{i+2,1} & & \end{array}$$

$k = 3$:

$$\begin{array}{ccccc} N_{i,0} & & & & \\ & N_{i,1}^{(1)} & & & \\ N_{i+1,0} & & N_{i,2}^{(2)} & & \\ & N_{i+1,1}^{(1)} & & N_{i,3}^{(3)} & \\ N_{i+2,0} & & N_{i+1,2}^{(2)} & & \\ & N_{i+2,1}^{(1)} & & & \\ N_{i+3,0} & & & & \end{array}$$

In words, the algorithm is:

1. compute and store the entire triangular table corresponding to $k = 0$;
2. to get the kth derivative, load the column of the table which contains the functions of degree $p - k$, and compute the remaining portion of the triangle.

Algorithm `A2.5` computes $N_{i,p}^{(k)}(u)$ for $k = 0, \ldots, n$, $n \leq p$. The kth derivative is returned in `ders[k]`.

```
ALGORITHM A2.5
  DersOneBasisFun(p,m,U,i,u,n,ders)
    { /*  Compute derivatives of basis function Nip  */
```



```
    /*  Input:  p,m,U,i,u,n  */
    /*  Output: ders  */
  if (u < U[i] || u >= U[i+p+1])   /* Local property */
    {
    for (k=0; k<=n; k++)  ders[k] = 0.0;
    return;
    }
  for (j=0; j<=p; j++)  /* Initialize zeroth-degree functs */
    if (u >= U[i+j] && u < U[i+j+1])  N[j][0] = 1.0;
      else                N[j][0] = 0.0;
  for (k=1; k<=p; k++)   /* Compute full triangular table */
    {
    if (N[0][k-1] == 0.0)  saved = 0.0;
      else    saved = ((u-U[i])*N[0][k-1])/(U[i+k]-U[i]);
    for (j=0; j<p-k+1; j++)
      {
      Uleft = U[i+j+1];
      Uright = U[i+j+k+1];
      if (N[j+1][k-1] == 0.0)
        {
        N[j][k] = saved;   saved = 0.0;
        }
        else
        {
        temp = N[j+1][k-1]/(Uright-Uleft);
        N[j][k] = saved+(Uright-u)*temp;
        saved = (u-Uleft)*temp;
        }
      }
    }
  ders[0] = N[0][p];   /* The function value */
  for (k=1; k<=n; k++)   /* Compute the derivatives */
    {
    for (j=0; j<=k; j++)   /* Load appropriate column */
      ND[j] = N[j][p-k];
    for (jj=1; jj<=k; jj++)   /* Compute table of width k */
      {
      if (ND[0] == 0.0)  saved = 0.0;
        else     saved = ND[0]/(U[i+p-k+jj]-U[i]);
      for (j=0; j<k-jj+1; j++)
        {
        Uleft = U[i+j+1];
        Uright = U[i+j+p+jj+1];
        if (ND[j+1] == 0.0)
          {
```

```
            ND[j] = (p-k+jj)*saved;    saved = 0.0;
            }
            else
            {
            temp = ND[j+1]/(Uright-Uleft);
            ND[j] = (p-k+jj)*(saved-temp);
            saved = temp;
            }
          }
        }
      ders[k] = ND[0];   /* kth derivative */
    }
  }
```

Finally, note that Algorithms A2.3 and A2.5 compute derivatives from the right if u is a knot. However, Eqs. (2.5), (2.9), (2.10), and others in this chapter could have been defined using intervals of the form $u \in (u_i, u_{i+1}]$. This would not change Algorithms A2.2 through A2.5. In other words, derivatives from the left can be found by simply having the span-finding algorithm use intervals of the form $(u_i, u_{i+1}]$, instead of $[u_i, u_{i+1})$. In the preceding example, with $p = 2$ and $U = \{0,0,0,1,2,3,4,4,5,5,5\}$, if $u = 2$ then span $i = 3$ yields derivatives from the left, and $i = 4$ yields derivatives from the right.

Exercises

2.1. Consider the linear and quadratic functions computed earlier and shown in Figures 2.5 and 2.6. Substitute $u = 5/2$ into the polynomial equations to obtain $N_{3,1}(5/2)$, $N_{4,1}(5/2)$, $N_{2,2}(5/2)$, $N_{3,2}(5/2)$, and $N_{4,2}(5/2)$. What do you notice about the sum of the two linear, and the sum of the three quadratic functions?

2.2. Consider the quadratic functions of Figure 2.6. Using the polynomial expressions for $N_{3,2}(u)$, evaluate the function and its first and second derivatives at $u = 2$ from both the left and right. Observe the continuity. Does Property P2.5 hold? Do the same with $N_{4,2}(u)$ at $u = 4$.

2.3. Let $U = \{0,0,0,0,1,2,3,4,4,5,5,5,5\}$. How does this change the degree 0, 1, and 2 functions of Figures 2.4–2.6? Compute and sketch the nine cubic basis functions associated with U.

2.4. Consider the function $N_{2,2}(u)$ of Figure 2.5, $N_{2,2}(u) = \frac{1}{2}u^2$ on $[0,1)$, $-\frac{3}{2} + 3u - u^2$ on $[1,2)$ and $\frac{1}{2}(3-u)^2$ on $[2,3)$. Use Eq. (2.10) to obtain the expressions for the first and second derivatives of $N_{2,2}(u)$.

2.5. Again consider $N_{2,2}(u)$ of Figure 2.5. Obtain the first derivatives of $N_{2,1}$ and $N_{3,1}$ by differentiating the polynomial expressions directly. Then use these, together with Eq. (2.11), to obtain $N'_{2,2}$.

2.6. Again let $p = 2$, $u = 5/2$, and $U = \{0,0,0,1,2,3,4,4,5,5,5\}$. Trace through Algorithm A2.2 by hand to find the values of the three nonzero basis functions. Trace through Algorithm A2.3 to find the first and second derivatives of the basis functions.

2.7. Use the same p and U as in Exercise 2.6, with $u = 2$. Trace through Algorithm A2.3 with $n = 1$, once with $i = 3$, and once with $i = 4$. Then differentiate the appropriate polynomial expressions for the $N_{j,2}$ given in Section 2.2, and evaluate the derivatives from the left and right at $u = 2$. Compare the results with what you obtained from Algorithm A2.3.

2.8. Using the same p and U as in Exercise 2.6, let $u = 4$. Trace through Algorithms A2.2 and A2.3 to convince yourself there are no problems with double knots.

2.9. With the same p and U as in Exercise 2.6, let $u = 5/2$. Trace through Algorithm A2.5 and compute the derivatives $N^{(k)}_{4,2}(5/2)$ for $k = 0, 1, 2$.

CHAPTER

THREE

B-spline Curves and Surfaces

3.1 Introduction

In this chapter we define nonrational B-spline curves and surfaces, study their properties, and derive expressions for their derivatives. For brevity we drop the word nonrational for the remainder of this chapter. The primary goal is to acquire an intuitive understanding of B-spline curves and surfaces, and to that end the reader should carefully study the many examples and figures given in this chapter. We also give algorithms for computing points and derivatives on B-spline curves and surfaces. The use of B-splines to define curves and surfaces for computer-aided geometric design was first proposed by Gordon and Riesenfeld [Gord74b; Ries73]. B-spline techniques are now covered in many books on curves and surfaces – see [DeBo78; Mort85; Bart87; Fari93; Yama88; Hosc93; Su89; Roge90; Beac91].

3.2 The Definition and Properties of B-spline Curves

A *pth-degree B-spline curve* is defined by

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i \qquad a \le u \le b \tag{3.1}$$

where the $\{\mathbf{P}_i\}$ are the *control points*, and the $\{N_{i,p}(u)\}$ are the pth-degree B-spline basis functions (Eq. [2.5]) defined on the nonperiodic (and nonuniform) knot vector

$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\}$$

$(m+1$ knots). Unless stated otherwise, we assume that $a = 0$ and $b = 1$. The polygon formed by the $\{\mathbf{P}_i\}$ is called the *control polygon*. Examples of B-spline curves (in some cases together with their corresponding basis functions) are shown in Figures 3.1–3.14.

Three steps are required to compute a point on a B-spline curve at a fixed u value:

1. find the knot span in which u lies (Algorithm `A2.1`);
2. compute the nonzero basis functions (Algorithm `A2.2`);
3. multiply the values of the nonzero basis functions with the corresponding control points.

Consider Example `Ex2.3` of Section 2.5, with $U = \{0,0,0,1,2,3,4,4,5,5,5\}$, $u = 5/2$, and $p = 2$. Then $u \in [u_4, u_5)$, and

$$N_{2,2}\left(\frac{5}{2}\right) = \frac{1}{8} \qquad N_{3,2}\left(\frac{5}{2}\right) = \frac{6}{8} \qquad N_{4,2}\left(\frac{5}{2}\right) = \frac{1}{8}$$

Multiplying with the control points yields

$$\mathbf{C}\left(\frac{5}{2}\right) = \frac{1}{8}\mathbf{P}_2 + \frac{6}{8}\mathbf{P}_3 + \frac{1}{8}\mathbf{P}_4$$

The algorithm follows.

```
ALGORITHM A3.1:
  CurvePoint(n,p,U,P,u,C)
    { /* Compute curve point  */
    /*  Input: n,p,U,P,u  */
    /*  Output: C  */
    span = FindSpan(n,p,u,U);
    BasisFuns(span,u,p,U,N);
    C = 0.0;
    for (i=0; i<=p; i++)
      C = C + N[i]*P[span-p+i];
    }
```

We now list a number of properties of B-spline curves. These properties follow from those given in Chapter 2 for the functions $N_{i,p}(u)$. Let $\mathbf{C}(u)$ be defined by Eq. (3.1).

`P3.1` If $n = p$ and $U = \{0,\ldots,0,1,\ldots,1\}$, then $\mathbf{C}(u)$ is a Bézier curve (Figure 3.1);

`P3.2` $\mathbf{C}(u)$ is a piecewise polynomial curve (since the $N_{i,p}(u)$ are piecewise polynomials); the degree, p, number of control points, $n+1$, and number of knots, $m+1$, are related by

$$m = n + p + 1 \tag{3.2}$$

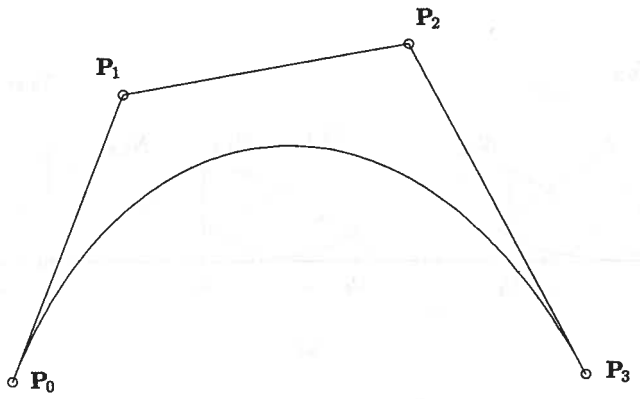


Figure 3.1. A cubic B-spline curve on $U = \{0,0,0,0,1,1,1,1\}$, i.e., a cubic Bézier curve.

(see Section 2.4). Figures 3.2 and 3.3 show basis functions and sections of the B-spline curves corresponding to the individual knot spans; in both figures the alternating solid/dashed segments correspond to the different polynomials (knot spans) defining the curve.

`P3.3` Endpoint interpolation: $\mathbf{C}(0) = \mathbf{P}_0$ and $\mathbf{C}(1) = \mathbf{P}_n$;

`P3.4` Affine invariance: an affine transformation is applied to the curve by applying it to the control points. Let $\mathbf{r}$ be a point in $\mathcal{E}^3$ (three-dimensional Euclidean space). An *affine transformation*, denoted by Φ, maps $\mathcal{E}^3$ into $\mathcal{E}^3$ and has the form

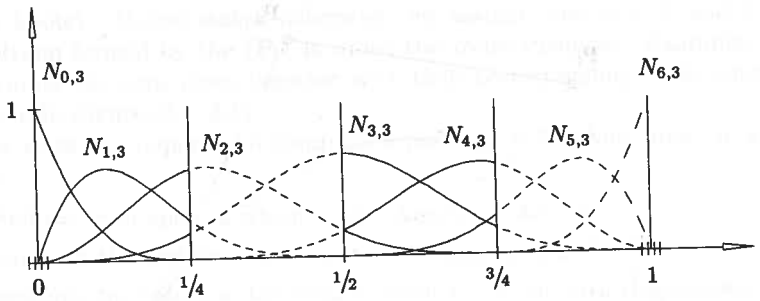
$$\Phi(\mathbf{r}) = A\mathbf{r} + \mathbf{v}$$

where A is a 3×3 matrix and $\mathbf{v}$ is a vector. Affine transformations include translations, rotations, scalings, and shears. The affine invariance property for B-spline curves follows from the partition of unity property of the $N_{i,p}(u)$. Thus, let $\mathbf{r} = \sum \alpha_i \mathbf{p}_i$, where $\mathbf{p}_i \in \mathcal{E}^3$ and $\sum \alpha_i = 1$. Then

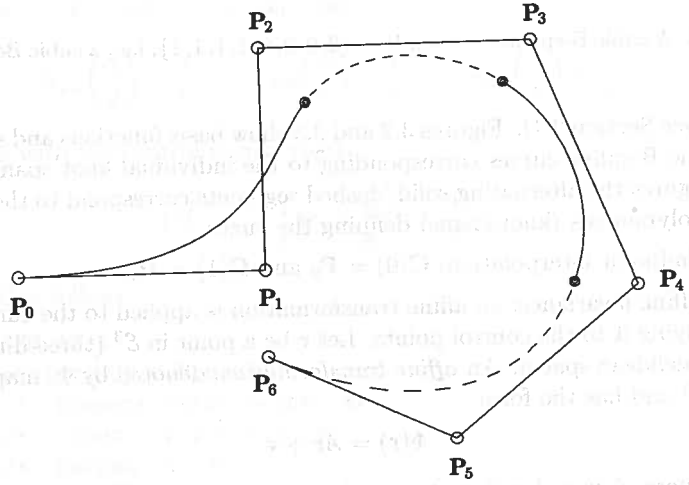
$$\Phi(\mathbf{r}) = \Phi\left(\sum \alpha_i \mathbf{p}_i\right) = A\left(\sum \alpha_i \mathbf{p}_i\right) + \mathbf{v} = \sum \alpha_i A\mathbf{p}_i + \sum \alpha_i \mathbf{v}$$

$$= \sum \alpha_i (A\mathbf{p}_i + \mathbf{v}) = \sum \alpha_i \Phi(\mathbf{p}_i)$$

`P3.5` Strong convex hull property: the curve is contained in the convex hull of its control polygon; in fact, if $u \in [u_i, u_{i+1})$, $p \le i < m-p-1$, then $\mathbf{C}(u)$ is in the convex hull of the control points $\mathbf{P}_{i-p},\ldots,\mathbf{P}_i$ (Figures 3.4, 3.5, and 3.6). This follows from the nonnegativity and partition of unity properties of the $N_{i,p}(u)$ (Properties `P2.3` and `P2.4`), and the property that $N_{j,p}(u) = 0$ for $j < i-p$ and $j > i$ when $u \in [u_i, u_{i+1})$ (Property `P2.2`). Figure 3.6 shows how to construct a quadratic curve containing a straight line segment. Since $\mathbf{P}_2$, $\mathbf{P}_3$, and $\mathbf{P}_4$ are colinear, the strong



(a)



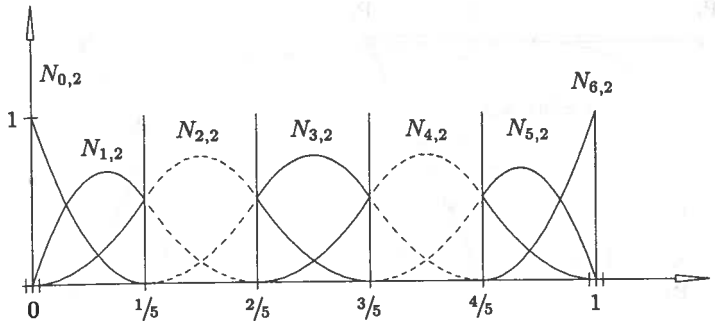
(b)

Figure 3.2. (a) Cubic basis functions $U = \{0,0,0,0,\frac{1}{4},\frac{1}{2},\frac{3}{4},1,1,1,1\}$; (b) a cubic curve using the basis functions of Figure 3.2a.

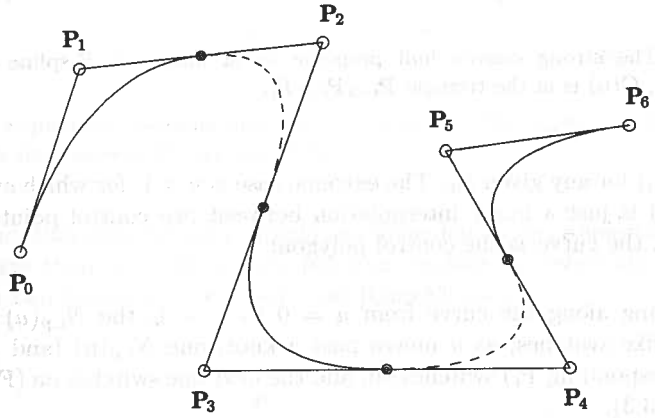
convex hull property forces the curve to be a straight line segment from $\mathbf{C}(2/5)$ to $\mathbf{C}(3/5)$;

P3.6 Local modification scheme: moving $\mathbf{P}_i$ changes $\mathbf{C}(u)$ only in the interval $[u_i, u_{i+p+1})$ (Figure 3.7). This follows from the fact that $N_{i,p}(u) = 0$ for $u \notin [u_i, u_{i+p+1})$ (Property P2.1).

P3.7 The control polygon represents a piecewise linear approximation to the curve; this approximation is improved by knot insertion or degree elevation (see Chapter 5). As a general rule, the lower the degree, the closer a B-spline curve follows its control polygon (see Figures 3.8 and 3.9). The curves of Figure 3.9 are defined using the same six control points, and the knot vectors



(a)



(b)

Figure 3.3. (a) Quadratic basis functions on $U = \{0,0,0,\frac{1}{5},\frac{2}{5},\frac{3}{5},\frac{4}{5},1,1,1\}$; (b) a quadratic curve using the basis functions of Figure 3.3a.

$$p = 1 : U = \left\{0,0,\frac{1}{5},\frac{2}{5},\frac{3}{5},\frac{4}{5},1,1\right\}$$

$$p = 2 : U = \left\{0,0,0,\frac{1}{4},\frac{1}{2},\frac{3}{4},1,1,1\right\}$$

$$p = 3 : U = \left\{0,0,0,0,\frac{1}{3},\frac{2}{3},1,1,1,1\right\}$$

$$p = 4 : U = \left\{0,0,0,0,0,\frac{1}{2},1,1,1,1,1\right\}$$

$$p = 5 : U = \{0,0,0,0,0,0,1,1,1,1,1,1\}$$

The reason for this phenomenon is intuitive: the lower the degree, the fewer the control points that are contributing to the computation of

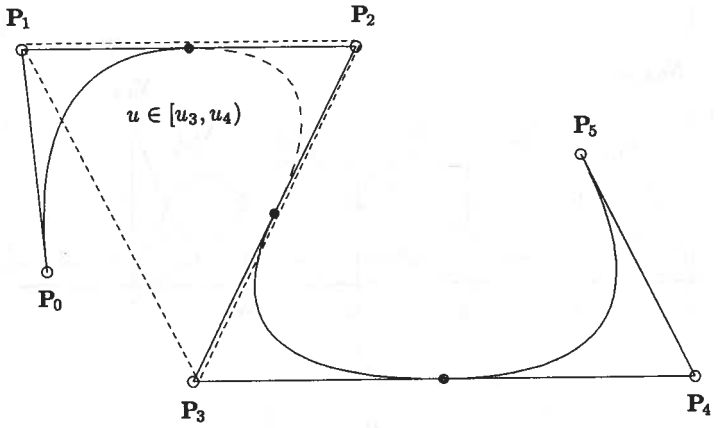


Figure 3.4. The strong convex hull property for a quadratic B-spline curve; for $u \in [u_i, u_{i+1})$, $\mathbf{C}(u)$ is in the triangle $\mathbf{P}_{i-2}\mathbf{P}_{i-1}\mathbf{P}_i$.

$\mathbf{C}(u_0)$ for any given u_0. The extreme case is $p = 1$, for which every point $\mathbf{C}(u)$ is just a linear interpolation between two control points. In this case, the curve *is* the control polygon;

P3.8 Moving along the curve from $u = 0$ to $u = 1$, the $N_{i,p}(u)$ functions act like switches; as u moves past a knot, one $N_{i,p}(u)$ (and hence the corresponding $\mathbf{P}_i$) switches off, and the next one switches on (Figures 3.2 and 3.3).

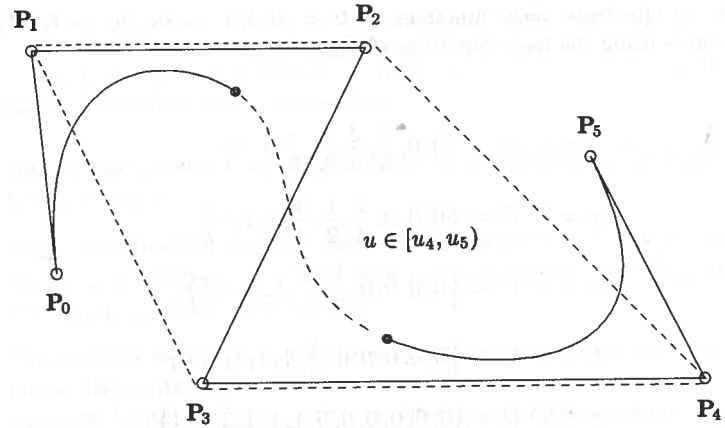


Figure 3.5. The strong convex hull property for a cubic B-spline curve; for $u \in [u_i, u_{i+1})$, $\mathbf{C}(u)$ is in the quadrilateral $\mathbf{P}_{i-3}\mathbf{P}_{i-2}\mathbf{P}_{i-1}\mathbf{P}_i$.

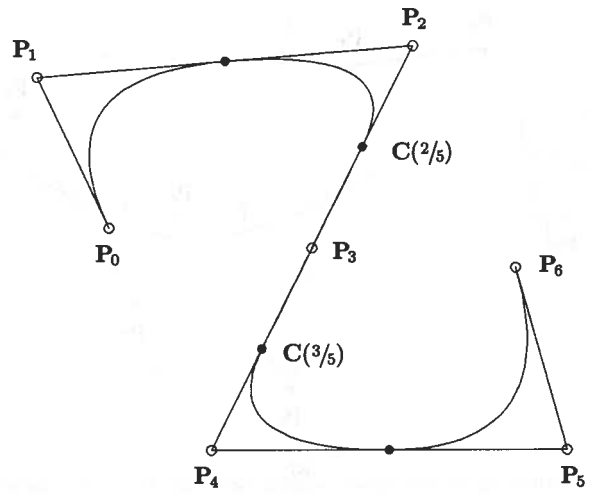


Figure 3.6. A quadratic B-spline curve on $U = \{0, 0, 0, 1/5, 2/5, 3/5, 4/5, 1, 1, 1\}$. The curve is a straight line between $\mathbf{C}(2/5)$ and $\mathbf{C}(3/5)$.

P3.9 Variation diminishing property: no plane has more intersections with the curve than with the control polygon (replace the word plane with line, for two-dimensional curves) – see [Lane83] for proof;

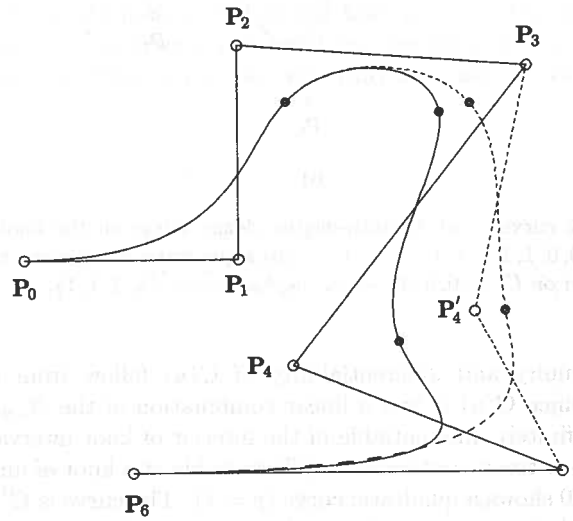
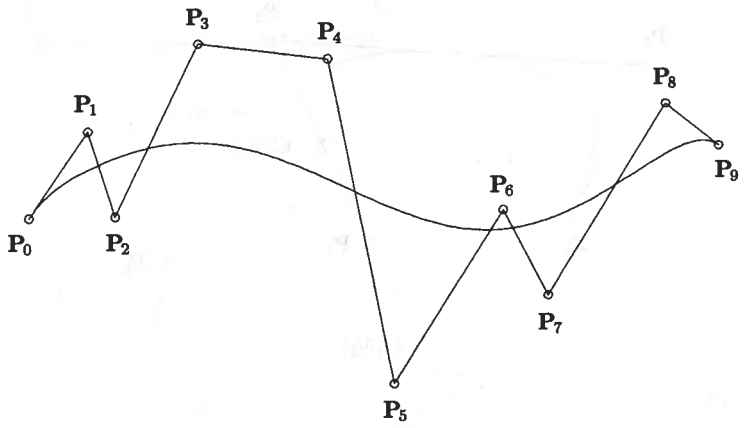
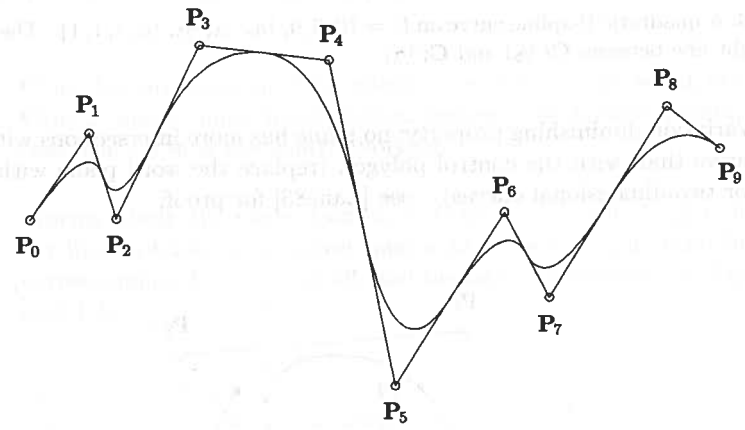


Figure 3.7. A cubic curve on $U = \{0, 0, 0, 0, 1/4, 1/2, 3/4, 1, 1, 1, 1\}$; moving $\mathbf{P}_4$ (to $\mathbf{P}'_4$) changes the curve in the interval $[1/4, 1)$.



(a)



(b)

Figure 3.8. B-spline curves. (a) A ninth-degree Bézier curve on the knot vector $U = \{0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1\}$; (b) a quadratic curve using the same control polygon defined on $U = \{0,0,0,\frac{1}{8},\frac{2}{8},\frac{3}{8},\frac{4}{8},\frac{5}{8},\frac{6}{8},\frac{7}{8},1,1,1\}$.

P3.10 The continuity and differentiability of $\mathbf{C}(u)$ follow from that of the $N_{i,p}(u)$ (since $\mathbf{C}(u)$ is just a linear combination of the $N_{i,p}(u)$). Thus, $\mathbf{C}(u)$ is infinitely differentiable in the interior of knot intervals, and it is at least $p-k$ times continuously differentiable at a knot of multiplicity k. Figure 3.10 shows a quadratic curve ($p = 2$). The curve is $\mathbf{C}^1$ continuous (the first derivative is continuous but the second is not) at all interior knots of multiplicity 1. At the double knot, $u = \frac{4}{5}$, $\mathbf{C}(u)$ is only $\mathbf{C}^0$ continuous, thus there is a cusp (a visual discontinuity). Figure 3.11 shows a

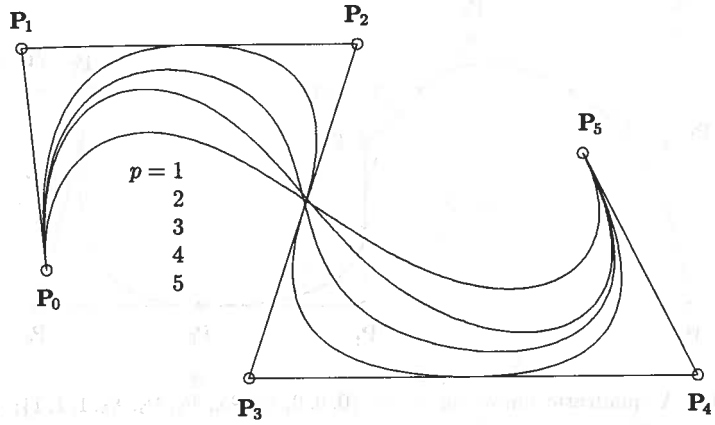


Figure 3.9. B-spline curves of different degree, using the same control polygon.

quadratic curve defined on the same knot vector. Hence, the two curves use the same basis functions, $N_{i,p}(u)$, for their definitions. But the curve of Figure 3.11 is $\mathbf{C}^1$ continuous at $u = \frac{4}{5}$; this is not obvious but can be seen using the derivative expression given in Section 3.3. This is simply a consequence of the fact that discontinuous functions can sometimes be combined in such a way that the result is continuous. Notice that $\mathbf{P}_4$, $\mathbf{P}_5$, and $\mathbf{P}_6$ are colinear, and length($\mathbf{P}_4\mathbf{P}_5$) = length($\mathbf{P}_5\mathbf{P}_6$). Figure 3.12 shows a cubic curve which is $\mathbf{C}^2$ continuous at $u = \frac{1}{4}$ and $u = \frac{1}{2}$, but only $\mathbf{C}^1$ continuous at the double knot $u = \frac{3}{4}$. The eye detects discontinuities in the second derivative but probably not in third and higher derivatives. Thus, cubics are generally adequate for visual purposes.

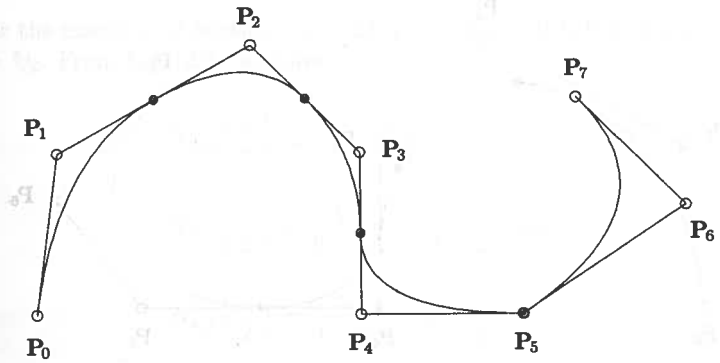


Figure 3.10. A quadratic curve on $U = \{0,0,0,\frac{1}{5},\frac{2}{5},\frac{3}{5},\frac{4}{5},\frac{4}{5},1,1,1\}$ with a cusp at $u = \frac{4}{5}$.

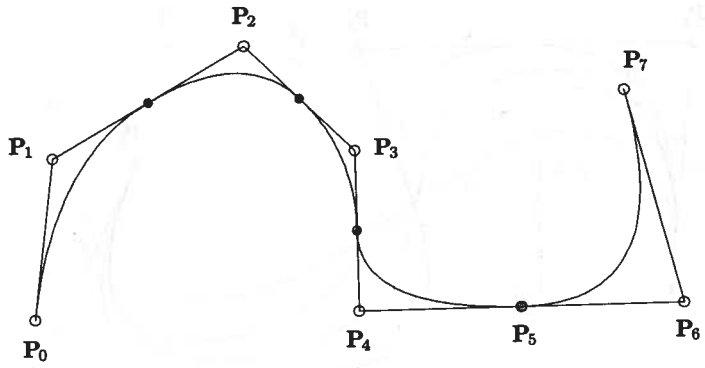


Figure 3.11. A quadratic curve on $U = \{0,0,0,\frac{1}{5},\frac{2}{5},\frac{3}{5},\frac{4}{5},\frac{4}{5},1,1,1\}$; the first derivative is continuous at $u = \frac{4}{5}$.

P3.11 It is possible (and sometimes useful) to use multiple (coincident) control points. Figure 3.13 shows a quadratic curve with a double control point, $\mathbf{P}_2 = \mathbf{P}_3$. The interesting portion of this curve lies between $\mathbf{C}(\frac{5}{2})$ and $\mathbf{C}(\frac{3}{4})$. Indeed, $\mathbf{C}(\frac{1}{2}) = \mathbf{P}_2 = \mathbf{P}_3$, and the curve segments between $\mathbf{C}(\frac{1}{4})$ and $\mathbf{C}(\frac{1}{2})$, and $\mathbf{C}(\frac{1}{2})$ and $\mathbf{C}(\frac{3}{4})$, are straight lines. This follows from Property P3.5, e.g., $\mathbf{C}(u)$ is in the convex hull of $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ (a line) if $u \in [\frac{1}{4}, \frac{1}{2})$. Furthermore, since the knot $u = \frac{1}{2}$ has multiplicity $= 1$, the curve must be $\mathbf{C}^1$ continuous there, even though it has a cusp (visual discontinuity). This is a result of the magnitude of the first derivative vector going to zero (continuously) at $u = \frac{1}{2}$. In the next section we see that the derivative at $u = \frac{1}{2}$ is proportional to the difference, $\mathbf{P}_3 - \mathbf{P}_2$. Figures 3.14a and 3.14b are cubic examples using the same

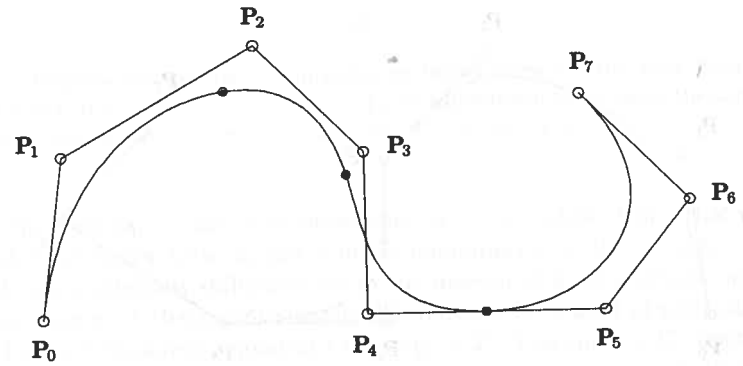


Figure 3.12. A cubic curve on $U = \{0,0,0,0,\frac{1}{4},\frac{1}{2},\frac{3}{4},\frac{3}{4},1,1,1,1\}$, $\mathbf{C}^2$ continuous at $u = \frac{1}{4}$ and $u = \frac{1}{2}$, and $\mathbf{C}^1$ continuous at $u = \frac{3}{4}$.

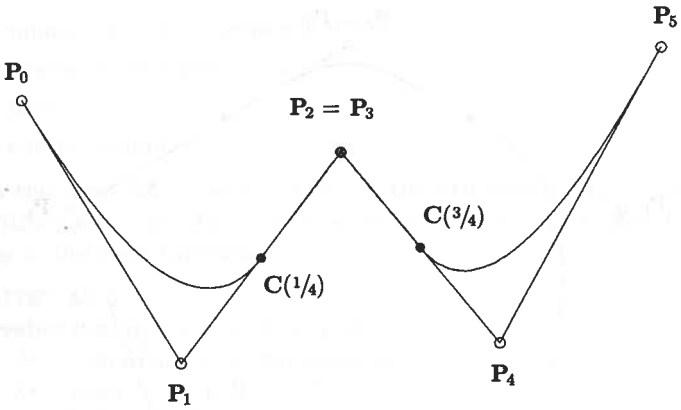


Figure 3.13. A quadratic curve on $U = \{0,0,0,\frac{1}{4},\frac{1}{2},\frac{3}{4},1,1,1\}$; $\mathbf{P}_2 = \mathbf{P}_3$ is a double control point.

control polygon, including the double control point, $\mathbf{P}_2 = \mathbf{P}_3$, but with different knot vectors.

3.3 The Derivatives of a B-spline Curve

Let $\mathbf{C}^{(k)}(u)$ denote the kth derivative of $\mathbf{C}(u)$. If u is fixed, we can obtain $\mathbf{C}^{(k)}(u)$ by computing the kth derivatives of the basis functions (see Eqs. [2.7], [2.9], and [2.10] and Algorithm A2.3). In particular

$$\mathbf{C}^{(k)}(u) = \sum_{i=0}^{n} N_{i,p}^{(k)}(u)\mathbf{P}_i \tag{3.3}$$

Consider the example of Section 2.5, with $p = 2$, $U = \{0,0,0,1,2,3,4,4,5,5,5\}$, and $u = \frac{5}{2}$. From Eq. (2.7) we have

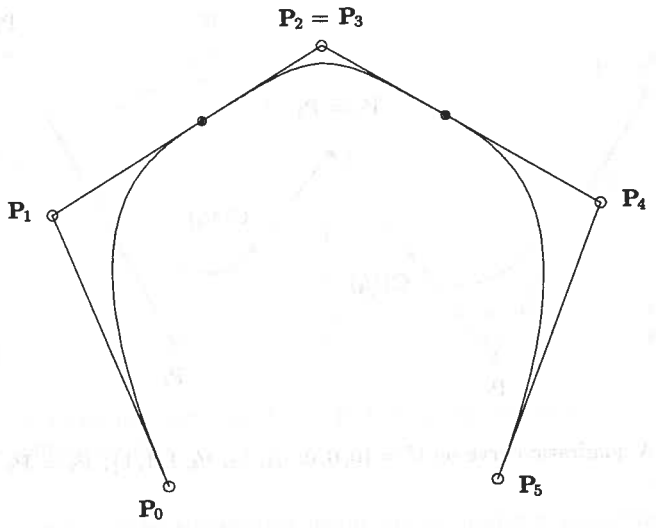
$$N'_{2,2}\left(\frac{5}{2}\right) = 0 - \frac{2}{3-1}\frac{1}{2} = -\frac{1}{2}$$

$$N'_{3,2}\left(\frac{5}{2}\right) = \frac{2}{3-1}\frac{1}{2} - \frac{2}{4-2}\frac{1}{2} = 0$$

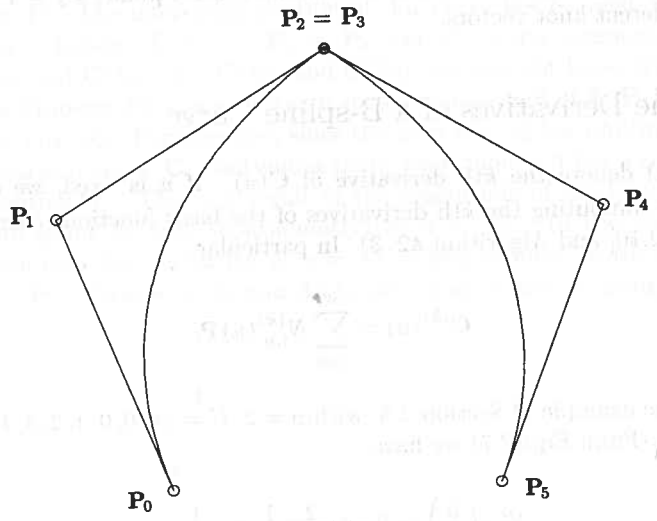
$$N'_{4,2}\left(\frac{5}{2}\right) = \frac{2}{4-2}\frac{1}{2} - 0 = \frac{1}{2}$$

It follows that

$$\mathbf{C}'\left(\frac{5}{2}\right) = -\frac{1}{2}\mathbf{P}_2 + \frac{1}{2}\mathbf{P}_4$$



(a)



(b)

Figure 3.14. Cubic curves with double control point $\mathbf{P}_2 = \mathbf{P}_3$. (a) $U = \{0,0,0,0,\frac{1}{4},\frac{3}{4},1,1,1,1\}$; (b) $U = \{0,0,0,0,\frac{1}{2},\frac{1}{2},1,1,1,1\}$.

An algorithm to compute the point on a B-spline curve and all derivatives up to and including the dth at a fixed u value follows. We allow $d > p$, although the derivatives are 0 in this case (for nonrational curves); these derivatives are necessary for rational curves. Input to the algorithm is `u,d`, and the B-spline curve, defined (throughout the remainder of this book) by

`n` : the number of control points is $n+1$;

`p` : the degree of the curve;

`U` : the knots;

`P` : the control points.

Output is the array `CK[]`, where `CK[k]` is the kth derivative, $0 \le k \le d$. We use Algorithms `A2.1` and `A2.3`. A local array, `nders[][]`, is used to store the derivatives of the basis functions.

```
ALGORITHM A3.2:
  CurveDerivsAlg1(n,p,U,P,u,d,CK)
    { /*  Compute curve derivatives  */
      /*  Input:  n,p,U,P,u,d  */
      /*  Output: CK  */
    du = min(d,p);
    for (k=p+1; k<=d; k++)   CK[k] = 0.0;
    span = FindSpan(n,p,u,U);
    DersBasisFuns(span,u,p,du,U,nders);
    for (k=0; k<=du; k++)
      {
      CK[k] = 0.0;
      for (j=0; j<=p; j++)
        CK[k] = CK[k] + nders[k][j]*P[span-p+j];
      }
    }
```

Now instead of fixing u, we want to formally differentiate the pth-degree B-spline curve,

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

defined on the knot vector

$$U = \{\underbrace{0,\ldots,0}_{p+1}, u_{p+1},\ldots,u_{m-p-1}, \underbrace{1,\ldots,1}_{p+1}\}$$

From Eqs. (3.3) and (2.7) we obtain

$$\begin{aligned}\mathbf{C}'(u) &= \sum_{i=0}^{n} N'_{i,p}(u)\mathbf{P}_i \\ &= \sum_{i=0}^{n} \left(\frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \right) \mathbf{P}_i \\ &= \left(p \sum_{i=-1}^{n-1} N_{i+1,p-1}(u) \frac{\mathbf{P}_{i+1}}{u_{i+p+1} - u_{i+1}} \right)\end{aligned}$$

$$-\left(p\sum_{i=0}^{n} N_{i+1,p-1}(u)\frac{\mathbf{P}_i}{u_{i+p+1}-u_{i+1}}\right)$$

$$=p\frac{N_{0,p-1}(u)\mathbf{P}_0}{u_p-u_0}+p\sum_{i=0}^{n-1}N_{i+1,p-1}(u)\frac{(\mathbf{P}_{i+1}-\mathbf{P}_i)}{u_{i+p+1}-u_{i+1}}-p\frac{N_{n+1,p-1}(u)\mathbf{P}_n}{u_{n+p+1}-u_{n+1}}$$

The first and last terms evaluate to $0/0$, which is 0 by definition. Thus

$$\mathbf{C}'(u)=p\sum_{i=0}^{n-1}N_{i+1,p-1}(u)\frac{(\mathbf{P}_{i+1}-\mathbf{P}_i)}{u_{i+p+1}-u_{i+1}}=\sum_{i=0}^{n-1}N_{i+1,p-1}(u)\,\mathbf{Q}_i$$

where
$$\mathbf{Q}_i=p\frac{\mathbf{P}_{i+1}-\mathbf{P}_i}{u_{i+p+1}-u_{i+1}} \tag{3.4}$$

Now let U' be the knot vector obtained by dropping the first and last knots from U, i.e.

$$U'=\{\underbrace{0,\ldots,0}_{p},u_{p+1},\ldots,u_{m-p-1},\underbrace{1,\ldots,1}_{p}\} \tag{3.5}$$

(U' has $m-1$ knots). Then it is easy to check that the function $N_{i+1,p-1}(u)$, computed on U, is equal to $N_{i,p-1}(u)$ computed on U'. Thus

$$\mathbf{C}'(u)=\sum_{i=0}^{n-1}N_{i,p-1}(u)\,\mathbf{Q}_i \tag{3.6}$$

where the $\mathbf{Q}_i$ are defined by Eq. (3.4), and the $N_{i,p-1}(u)$ are computed on U'. Hence, $\mathbf{C}'(u)$ is a $(p-1)$th-degree B-spline curve.

Examples

Ex3.1 Let $\mathbf{C}(u)=\sum_{i=0}^{4}N_{i,2}(u)\mathbf{P}_i$ be a quadratic curve defined on

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

Then $U'=\{0,0,2/5,3/5,1,1\}$ and $\mathbf{C}'(u)=\sum_{i=0}^{3}N_{i,1}(u)\,\mathbf{Q}_i$, where

$$\mathbf{Q}_0=\frac{2(\mathbf{P}_1-\mathbf{P}_0)}{\frac{1}{3}-0}=6(\mathbf{P}_1-\mathbf{P}_0)$$

$$\mathbf{Q}_1=\frac{2(\mathbf{P}_2-\mathbf{P}_1)}{\frac{3}{4}-0}=\frac{8}{3}(\mathbf{P}_2-\mathbf{P}_1)$$

$$\mathbf{Q}_2=\frac{2(\mathbf{P}_3-\mathbf{P}_2)}{1-\frac{1}{3}}=3(\mathbf{P}_3-\mathbf{P}_2)$$

$$\mathbf{Q}_3=\frac{2(\mathbf{P}_4-\mathbf{P}_3)}{1-\frac{3}{4}}=8(\mathbf{P}_4-\mathbf{P}_3)$$

$\mathbf{C}(u)$ and $\mathbf{C}'(u)$ are shown in Figures 3.15a and 3.15b, respectively.

Ex3.2 Let $\mathbf{C}(u)=\sum_{i=0}^{6}N_{i,3}(u)\mathbf{P}_i$ be a cubic curve defined on

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

Then $U'=\{0,0,0,2/5,3/5,3/5,1,1,1\}$ and $\mathbf{C}'(u)=\sum_{i=0}^{5}N_{i,2}(u)\,\mathbf{Q}_i$,

where

$$\mathbf{Q}_0=\frac{3(\mathbf{P}_1-\mathbf{P}_0)}{\frac{1}{3}-0}=9(\mathbf{P}_1-\mathbf{P}_0)$$

$$\mathbf{Q}_1=\frac{3(\mathbf{P}_2-\mathbf{P}_1)}{\frac{2}{3}-0}=\frac{9}{2}(\mathbf{P}_2-\mathbf{P}_1)$$

$$\mathbf{Q}_2=\frac{3(\mathbf{P}_3-\mathbf{P}_2)}{\frac{2}{3}-0}=\frac{9}{2}(\mathbf{P}_3-\mathbf{P}_2)$$

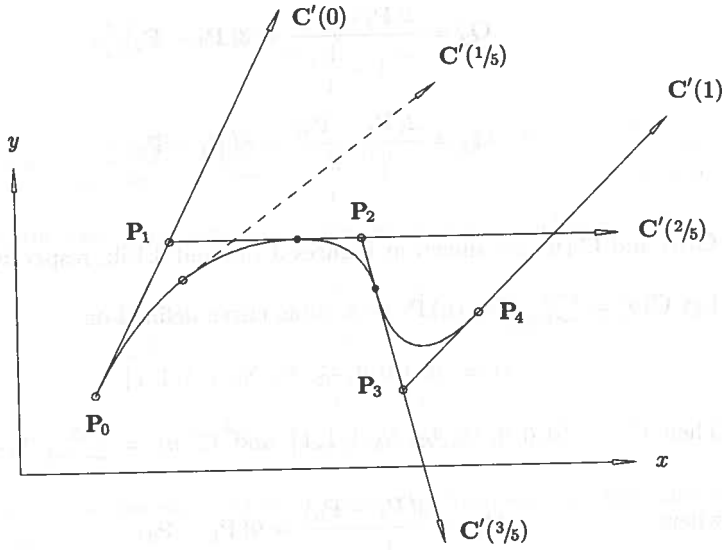
$$\mathbf{Q}_3=\frac{3(\mathbf{P}_4-\mathbf{P}_3)}{1-\frac{1}{3}}=\frac{9}{2}(\mathbf{P}_4-\mathbf{P}_3)$$

$$\mathbf{Q}_4=\frac{3(\mathbf{P}_5-\mathbf{P}_4)}{1-\frac{2}{3}}=9(\mathbf{P}_5-\mathbf{P}_4)$$

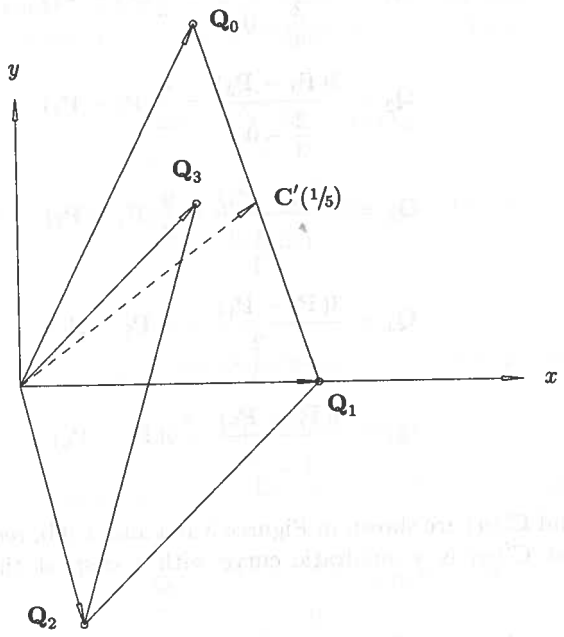
$$\mathbf{Q}_5=\frac{3(\mathbf{P}_6-\mathbf{P}_5)}{1-\frac{2}{3}}=9(\mathbf{P}_6-\mathbf{P}_5)$$

$\mathbf{C}(u)$ and $\mathbf{C}'(u)$ are shown in Figures 3.16a and 3.16b, respectively. Notice that $\mathbf{C}'(u)$ is a quadratic curve with a cusp at the double knot $u=3/5$.

Ex3.3 Recalling that a pth-degree Bézier curve is a B-spline curve on $U=\{0,\ldots,0,1,\ldots,1\}$ (no interior knots), Eq. (3.4) reduces to $\mathbf{Q}_i=p(\mathbf{P}_{i+1}-\mathbf{P}_i)$ for $0\le i\le n-1$. Since $n=p$ and $N_{i,p-1}(u)=B_{i,n-1}(u)$, the Bernstein polynomials, Eq. (3.6) is equivalent to Eq. (1.9).

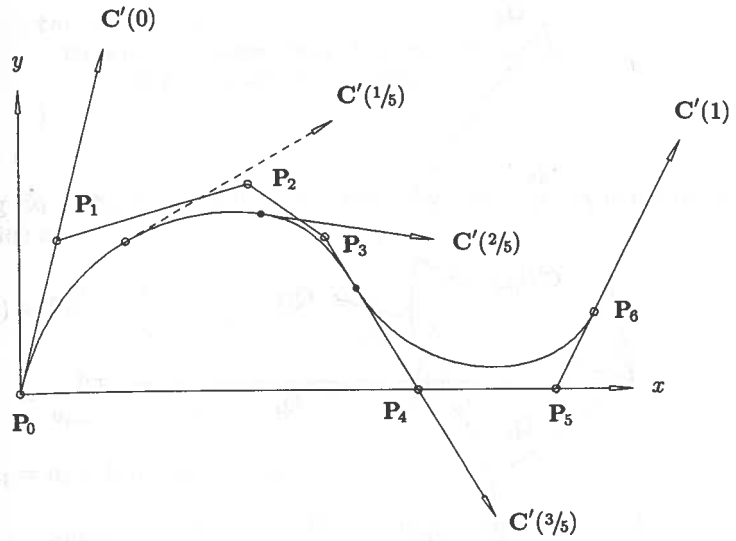


(a)



(b)

Figure 3.15. (a) A quadratic curve on $U = \{0,0,0,\ 2/5,\ 3/5,\ 1,1,1\}$; (b) the derivative of the curve is a first-degree B-spline curve on $U' = \{0,0,\ 2/5,\ 3/5,\ 1,1\}$.



(a)

Figure 3.16. (a) A cubic curve on $U = \{0,0,0,0,\ 2/5,\ 3/5,\ 3/5,1,1,1,1\}$; (b) the quadratic derivative curve on $U' = \{0,0,0,\ 2/5,\ 3/5,\ 3/5,1,1,1\}$.

The first derivatives at the endpoints of a B-spline curve are given by

$$\mathbf{C}'(0) = \mathbf{Q}_0 = \frac{p}{u_{p+1}}(\mathbf{P}_1 - \mathbf{P}_0)$$

$$\mathbf{C}'(1) = \mathbf{Q}_{n-1} = \frac{p}{1 - u_{m-p-1}}(\mathbf{P}_n - \mathbf{P}_{n-1}) \tag{3.7}$$

(see Examples Ex3.1 and Ex3.2, and Figures 3.15(a) and (b) and Figures 3.16 (a) and (b)). Note that in Figures 3.15b and 3.16b the derivative vectors and control point differences are scaled down for better visualization, by $1/2$ and by $1/3$, respectively.

Since $\mathbf{C}'(u)$ is a B-spline curve, we apply Eqs. (3.4) through (3.6) recursively to obtain higher derivatives. Letting $\mathbf{P}_i^{(0)} = \mathbf{P}_i$, we write

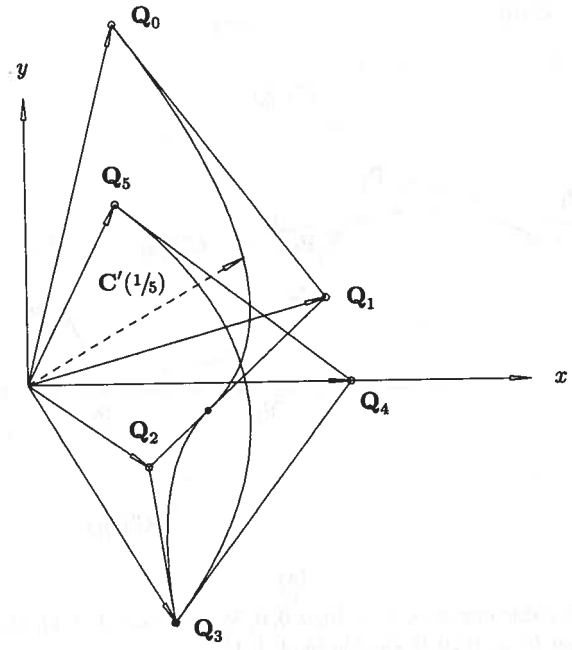
$$\mathbf{C}(u) = \mathbf{C}^{(0)}(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i^{(0)}$$

Then

$$\mathbf{C}^{(k)}(u) = \sum_{i=0}^{n-k} N_{i,p-k}(u)\mathbf{P}_i^{(k)} \tag{3.8}$$

with

$$\mathbf{P}_i^{(k)} = \begin{cases} \mathbf{P}_i & k = 0 \\ \dfrac{p-k+1}{u_{i+p+1} - u_{i+k}}\left(\mathbf{P}_{i+1}^{(k-1)} - \mathbf{P}_i^{(k-1)}\right) & k > 0 \end{cases}$$



(b)

Figure 3.16. (*Continued.*)

and $$U^{(k)} = \{\underbrace{0,\ldots,0}_{p-k+1}, u_{p+1},\ldots,u_{m-p-1}, \underbrace{1,\ldots,1}_{p-k+1}\}$$

Algorithm A3.3 is a nonrecursive implementation of Eq. (3.8). It computes the control points of all derivative curves up to and including the dth derivative ($d \le p$). On output, PK[k][i] is the ith control point of the kth derivative curve, where $0 \le k \le d$ and $r_1 \le i \le r_2 - k$. If $r_1 = 0$ and $r_2 = n$, all control points are computed.

```
ALGORITHM A3.3
  CurveDerivCpts(n,p,U,P,d,r1,r2,PK)
    { /*  Compute control points of curve derivatives  */
      /*  Input:  n,p,U,P,d,r1,r2  */
      /*  Output: PK  */
    r = r2-r1;
    for (i=0; i<=r; i++)
      PK[0][i] = P[r1+i];
    for (k=1; k<=d; k++)
      {
      tmp = p-k+1;
      for (i=0; i<=r-k; i++)
        PK[k][i] = tmp*(PK[k-1][i+1]-PK[k-1][i])/
          (U[r1+i+p+1]-U[r1+i+k]);
      }
    }
```


Using Eq. (3.8), we compute the second derivative at the endpoint, $u = 0$, of a B-spline curve ($p > 1$)

$$\mathbf{C}^{(2)}(0) = \mathbf{P}_0^{(2)} = \frac{p-2+1}{u_{p+1}-u_2}\left(\mathbf{P}_1^{(1)} - \mathbf{P}_0^{(1)}\right)$$

$$= \frac{p-1}{u_{p+1}-u_2}\left[\frac{p}{u_{p+2}-u_2}\left(\mathbf{P}_2^{(0)} - \mathbf{P}_1^{(0)}\right) - \frac{p}{u_{p+1}-u_1}\left(\mathbf{P}_1^{(0)} - \mathbf{P}_0^{(0)}\right)\right]$$

From $u_1 = u_2 = 0$ it follows that

$$\mathbf{C}^{(2)}(0) = \frac{p(p-1)}{u_{p+1}}\left[\frac{\mathbf{P}_0}{u_{p+1}} - \frac{(u_{p+1}+u_{p+2})\mathbf{P}_1}{u_{p+1}u_{p+2}} + \frac{\mathbf{P}_2}{u_{p+2}}\right] \tag{3.9}$$

Analogously,

$$\mathbf{C}^{(2)}(1) = \frac{p(p-1)}{1-u_{m-p-1}} \times$$

$$\left[\frac{\mathbf{P}_n}{1-u_{m-p-1}} - \frac{(2-u_{m-p-1}-u_{m-p-2})\mathbf{P}_{n-1}}{(1-u_{m-p-1})(1-u_{m-p-2})} + \frac{\mathbf{P}_{n-2}}{1-u_{m-p-2}}\right] \tag{3.10}$$

Notice that for Bézier curves these equations reduce to the corresponding expressions of Eq. (1.10). Figure 3.17 shows the quadratic curve of Figure 3.15a with the vectors $\mathbf{C}^{(2)}(0)$ and $\mathbf{C}^{(2)}(1)$. $\mathbf{C}^{(2)}(u)$ is a piecewise zeroth-degree curve, i.e., it is a constant (but different) vector on each of the three intervals $[0, 2/5)$, $[2/5, 3/5)$, and $[3/5, 1]$.

We close this section with another algorithm to compute the point on a B-spline curve and all derivatives up to and including the dth derivative at a fixed u value (compare with Algorithm A3.2). The algorithm is based on Eq. (3.8) and Algorithm A3.3. We assume a routine, AllBasisFuns, which is a simple modification of BasisFuns (Algorithm A2.2), to return all nonzero basis functions of all degrees from 0 up to p. In particular, N[j][i] is the value of the ith-degree basis function, $N_{\text{span}-i+j,i}(u)$, where $0 \le i \le p$ and $0 \le j \le i$.

```
ALGORITHM A3.4
  CurveDerivsAlg2(n,p,U,P,u,d,CK)
    {/*  Compute curve derivatives  */
      /*  Input:  n,p,U,P,u,d  */
      /*  Output: CK  */
    du = min(d,p);
```

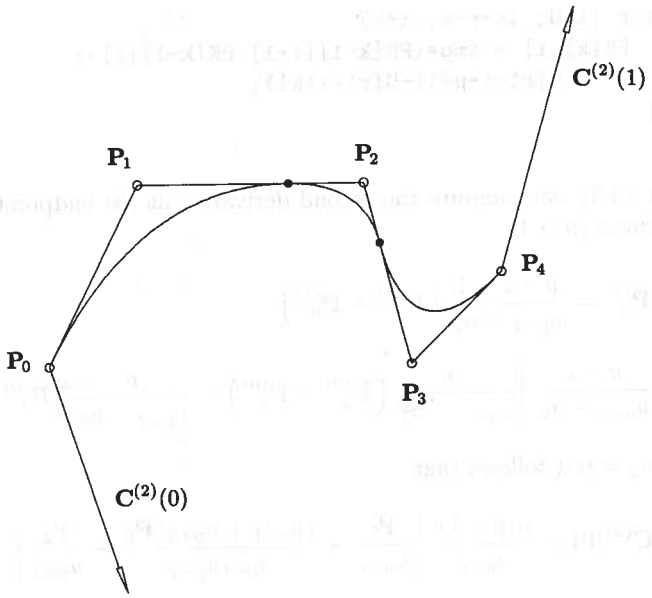


Figure 3.17. The second derivatives at the endpoints of the curve of Figure 3.15a.

```
for (k=p+1; k<=d; k++)   CK[k] = 0.0;
span = FindSpan(n,p,u,U);
AllBasisFuns(span,u,p,U,N);
CurveDerivCpts(n,p,U,P,du,span-p,span,PK);
for (k=0; k<=du; k++)
  {
  CK[k] = 0.0;
  for (j=0; j<=p-k; j++)
    CK[k] = CK[k] + N[j][p-k]*PK[k][j];
  }
}
```

Figure 3.18 shows a cubic curve with first, second, and third derivatives computed at $u = 2/5$. (The derivatives are scaled down by $2/5$.)

3.4 Definition and Properties of B-spline Surfaces

A B-spline surface is obtained by taking a bidirectional net of control points, two knot vectors, and the products of the univariate B-spline functions

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\mathbf{P}_{i,j} \tag{3.11}$$

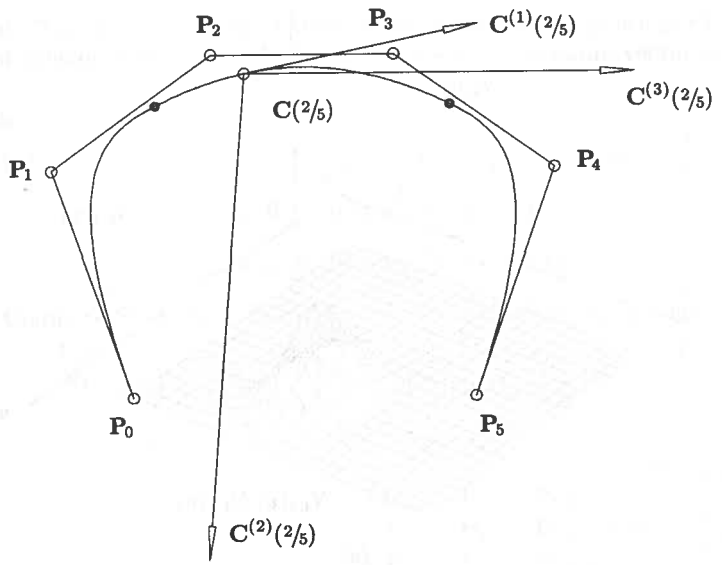


Figure 3.18. A cubic curve on $U = \{0,0,0,0,1/4,3/4,1,1,1,1\}$ with first, second, and third derivatives computed at $u = 2/5$.

with

$$U = \{\underbrace{0,\ldots,0}_{p+1}, u_{p+1},\ldots,u_{r-p-1}, \underbrace{1,\ldots,1}_{p+1}\}$$

$$V = \{\underbrace{0,\ldots,0}_{q+1}, v_{q+1},\ldots,v_{s-q-1}, \underbrace{1,\ldots,1}_{q+1}\}$$

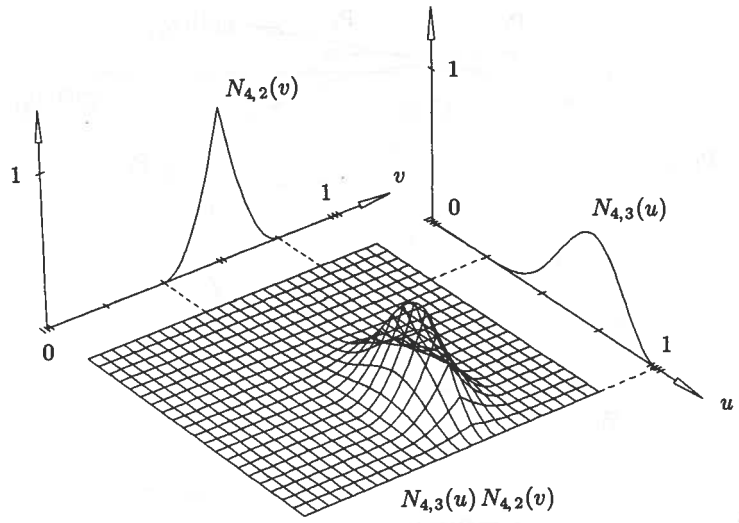
U has $r+1$ knots, and V has $s+1$. Equation (3.2) takes the form

$$r = n+p+1 \quad \text{and} \quad s = m+q+1 \tag{3.12}$$

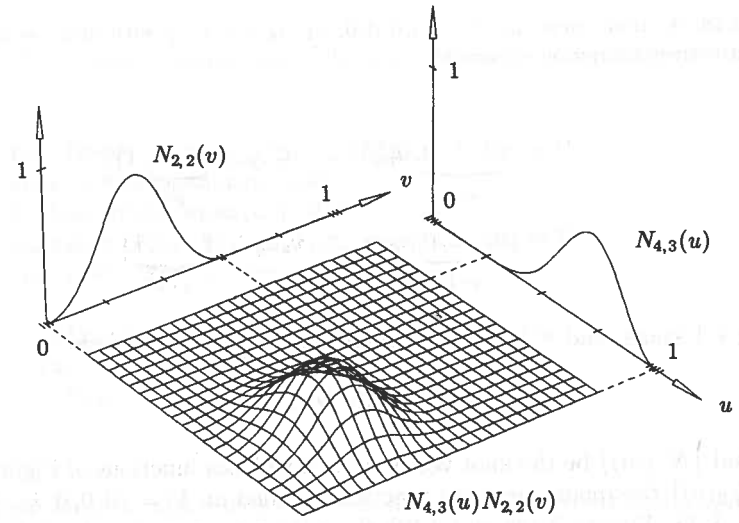
Let U and $\{N_{i,3}(u)\}$ be the knot vector and cubic basis functions of Figure 3.2a, and $\{N_{j,2}(v)\}$ the quadratic basis functions defined on $V = \{0,0,0,1/5,2/5,3/5,3/5,4/5,1,1,1\}$. Figures 3.19a and 3.19b show the tensor product basis functions $N_{4,3}(u)N_{4,2}(v)$ and $N_{4,3}(u)N_{2,2}(v)$, respectively. Figures 3.20–3.25 show examples of B-spline surfaces.

Five steps are required to compute a point on a B-spline surface at fixed (u,v) parameter values:

1. find the knot span in which u lies, say $u \in [u_i, u_{i+1})$ (Algorithm `A2.1`);
2. compute the nonzero basis functions $N_{i-p,p}(u),\ldots,N_{i,p}(u)$ (`A2.2`);
3. find the knot span in which v lies, say $v \in [v_j, v_{j+1})$ (`A2.1`);
4. compute the nonzero basis functions $N_{j-q,q}(v),\ldots,N_{j,q}(v)$ (`A2.2`);



(a)



(b)

Figure 3.19. Cubic × quadratic basis functions. (a) $N_{4,3}(u)N_{4,2}(v)$; (b) $N_{4,3}(u)N_{2,2}(v)$; $U = \{0,0,0,0,\frac{1}{4},\frac{1}{2},\frac{3}{4},1,1,1,1\}$ and $V = \{0,0,0,\frac{1}{5},\frac{2}{5},\frac{3}{5},\frac{3}{5},\frac{4}{5},1,1,1\}$.

5. multiply the values of the nonzero basis functions with the corresponding control points.

The last step takes the form

$$\mathbf{S}(u,v) = [\,N_{k,p}(u)\,]^T\,[\,\mathbf{P}_{k,l}\,]\,[\,N_{l,q}(v)\,] \qquad i-p \le k \le i,\;\; j-q \le l \le j \qquad (3.13)$$

Note that $[\,N_{k,p}(u)\,]^T$ is a $1\times(p+1)$ row vector of scalars, $[\,\mathbf{P}_{k,l}\,]$ is a $(p+1)\times(q+1)$ matrix of control points, and $[\,N_{l,q}(v)\,]$ is a $(q+1)\times 1$ column vector of scalars.

Example

Ex3.4 Let $p = q = 2$ and $\sum_{i=0}^{4}\sum_{j=0}^{5} N_{i,2}(u)N_{j,2}(v)\,\mathbf{P}_{i,j}$, with

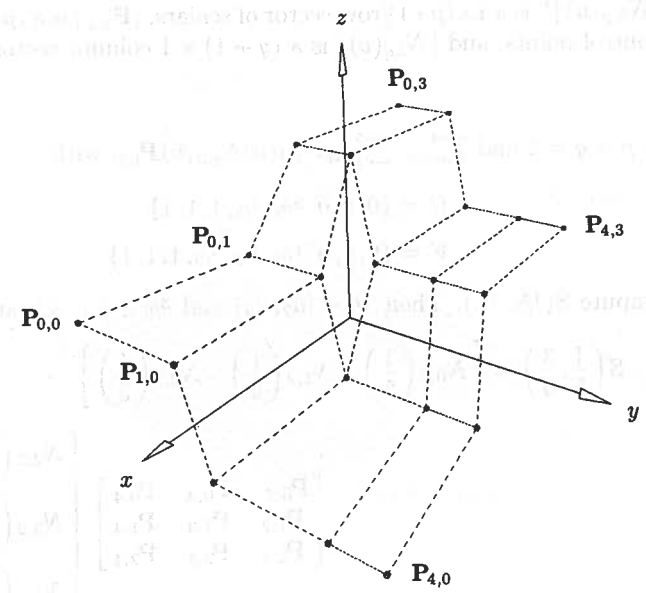
$$U = \{0,0,0,\tfrac{2}{5},\tfrac{3}{5},1,1,1\}$$
$$V = \{0,0,0,\tfrac{1}{5},\tfrac{1}{2},\tfrac{4}{5},1,1,1\}$$

Compute $\mathbf{S}(\frac{1}{5},\frac{3}{5})$. Then $\frac{1}{5} \in [u_2,u_3)$ and $\frac{3}{5} \in [v_4,v_5)$, and

$$\mathbf{S}\left(\frac{1}{5},\frac{3}{5}\right) = \begin{bmatrix} N_{0,2}\left(\frac{1}{5}\right) & N_{1,2}\left(\frac{1}{5}\right) & N_{2,2}\left(\frac{1}{5}\right)\end{bmatrix} \times \begin{bmatrix} \mathbf{P}_{0,2} & \mathbf{P}_{0,3} & \mathbf{P}_{0,4} \\ \mathbf{P}_{1,2} & \mathbf{P}_{1,3} & \mathbf{P}_{1,4} \\ \mathbf{P}_{2,2} & \mathbf{P}_{2,3} & \mathbf{P}_{2,4}\end{bmatrix} \begin{bmatrix} N_{2,2}\left(\frac{3}{5}\right) \\ N_{3,2}\left(\frac{3}{5}\right) \\ N_{4,2}\left(\frac{3}{5}\right)\end{bmatrix}$$

Algorithm A3.5 computes the point on a B-spline surface at fixed (u,v) values. For efficiency, it uses a local array, `temp[]`, to store the vector/matrix product, $[\,N_{k,p}(u)\,]^T\,[\,\mathbf{P}_{k,l}\,]$. The resulting vector of points (in `temp[]`) is then multiplied with the vector $[\,N_{l,q}(v)\,]$.

```
ALGORITHM A3.5
  SurfacePoint(n,p,U,m,q,V,P,u,v,S)
  { /*  Compute surface point  */
    /*  Input:  n,p,U,m,q,V,P,u,v  */
    /*  Output: S  */
   uspan = FindSpan(n,p,u,U);
   BasisFuns(uspan,u,p,U,Nu);
   vspan = FindSpan(m,q,v,V);
   BasisFuns(vspan,v,q,V,Nv);
   for (l=0; l<=q; l++)
    {
    temp[l] = 0.0;
    for (k=0; k<=p; k++)
      temp[l] = temp[l] + Nu[k]*P[uspan-p+k][vspan-q+l];
    }
   S = 0.0;
   for (l=0; l<=q; l++)
    S = S + Nv[l]*temp[l];
  }
```



(a)

Figure 3.20. A B-spline surface. (a) The control net; (b) the surface.

The properties of the tensor product basis functions follow from the corresponding properties of the univariate basis functions listed in Chapter 2.

P3.12 Nonnegativity: $N_{i,p}(u)N_{j,q}(v) \geq 0$ for all i, j, p, q, u, v;

P3.13 Partition of unity: $\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v) = 1$ for all $(u,v) \in [0,1] \times [0,1]$;

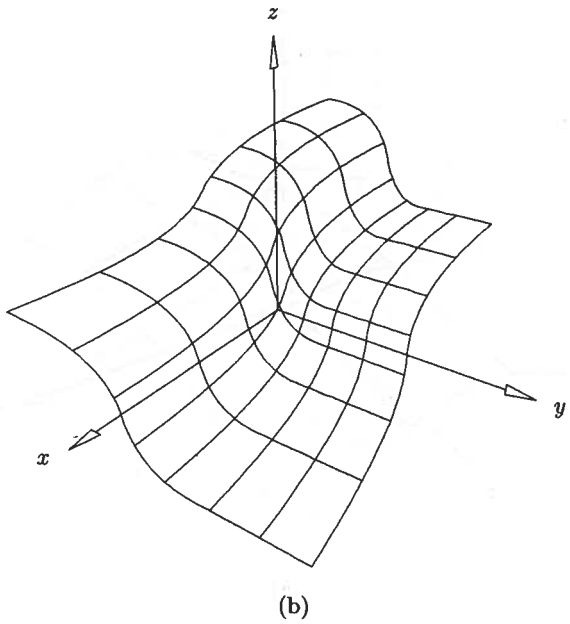
P3.14 If $n = p$, $m = q$, $U = \{0,\ldots,0,1,\ldots,1\}$, and $V = \{0,\ldots,0,1,\ldots,1\}$, then $N_{i,p}(u)N_{j,q}(v) = B_{i,n}(u)B_{j,m}(v)$ for all i, j; that is, products of B-spline functions degenerate to products of Bernstein polynomials;

P3.15 $N_{i,p}(u)N_{j,q}(v) = 0$ if (u,v) is outside the rectangle $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$ (see Figures 3.19a and 3.19b);

P3.16 In any given rectangle, $[u_{i_0}, u_{i_0+1}) \times [v_{j_0}, v_{j_0+1})$, at most $(p+1)(q+1)$ basis functions are nonzero, in particular the $N_{i,p}(u)N_{j,q}(v)$ for $i_0 - p \leq i \leq i_0$ and $j_0 - q \leq j \leq j_0$;

P3.17 If $p > 0$ and $q > 0$, then $N_{i,p}(u)N_{j,q}(v)$ attains exactly one maximum value (see Figures 3.19a and 3.19b);

P3.18 Interior to the rectangles formed by the u and v knot lines, where the function is a bivariate polynomial, all partial derivatives of $N_{i,p}(u)$ $N_{j,q}(v)$ exist; at a u knot (v knot) it is $p-k$ ($q-k$) times differentiable in the u (v) direction, where k is the multiplicity of the knot. In



(b)

Figure 3.20. (*Continued.*)

Figure 3.19a the first partial derivative of $N_{4,3}(u)N_{4,2}(v)$ with respect to v is discontinuous along the knot line $v = 3/5$ where $N_{4,2}(v)$ has a cusp. The second partial derivative with respect to u is everywhere continuous, because $N_{4,3}(u)$ is $\mathbf{C}^2$ continuous.

B-spline surfaces have the following properties:

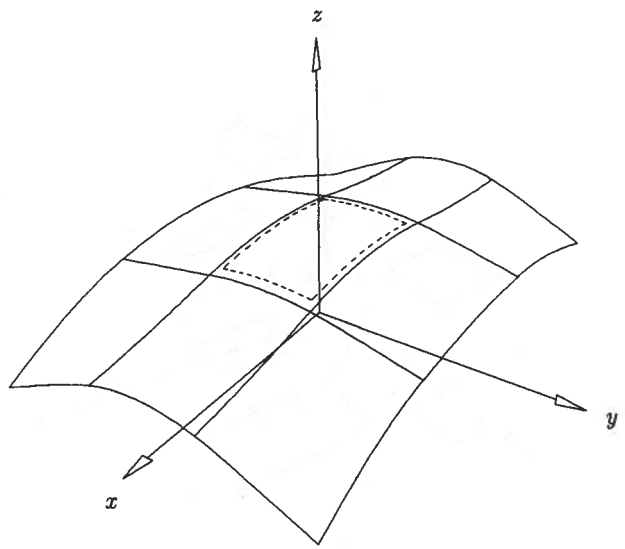
P3.19 If $n = p$, $m = q$, $U = \{0,\ldots,0,1,\ldots,1\}$, and $V = \{0,\ldots,0,1,\ldots,1\}$, then $\mathbf{S}(u,v)$ is a Bézier surface; this follows from P3.14;

P3.20 The surface interpolates the four corner control points: $\mathbf{S}(0,0) = \mathbf{P}_{0,0}$, $\mathbf{S}(1,0) = \mathbf{P}_{n,0}$, $\mathbf{S}(0,1) = \mathbf{P}_{0,m}$, and $\mathbf{S}(1,1) = \mathbf{P}_{n,m}$ (see Figures 3.20 through 3.25); this follows from P3.13 and the identity

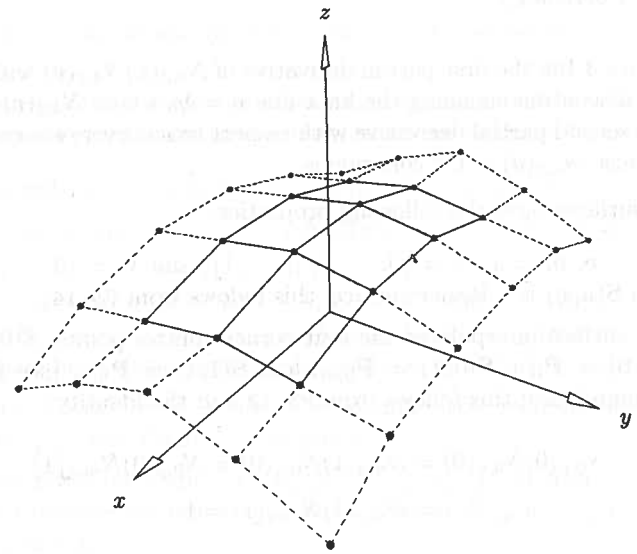
$$\begin{aligned} N_{0,p}(0)N_{0,q}(0) &= N_{n,p}(1)N_{0,q}(0) = N_{0,p}(0)N_{m,q}(1) \\ &= N_{n,p}(1)N_{m,q}(1) = 1 \end{aligned}$$

P3.21 Affine invariance: an affine transformation is applied to the surface by applying it to the control points; this follows from P3.13;

P3.22 Strong convex hull property: if $(u,v) \in [u_{i_0}, u_{i_0+1}) \times [v_{j_0}, v_{j_0+1})$, then $\mathbf{S}(u,v)$ is in the convex hull of the control points $\mathbf{P}_{i,j}$, $i_0 - p \leq i \leq i_0$ and $j_0 - q \leq j \leq j_0$ (see Figures 3.21); this follows from P3.12, P3.13, and P3.16;



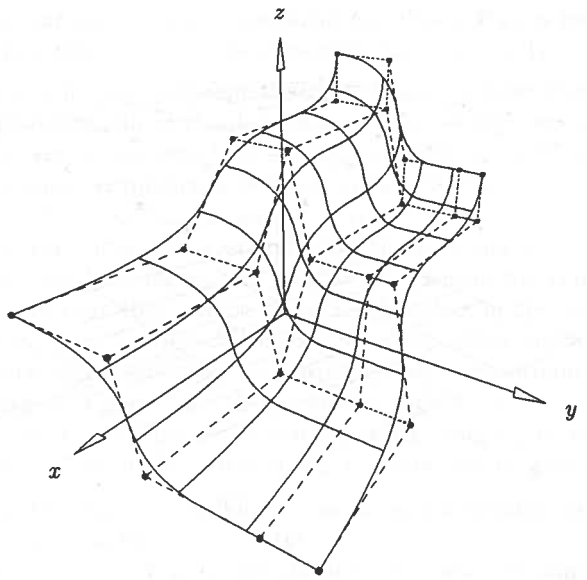
(a)



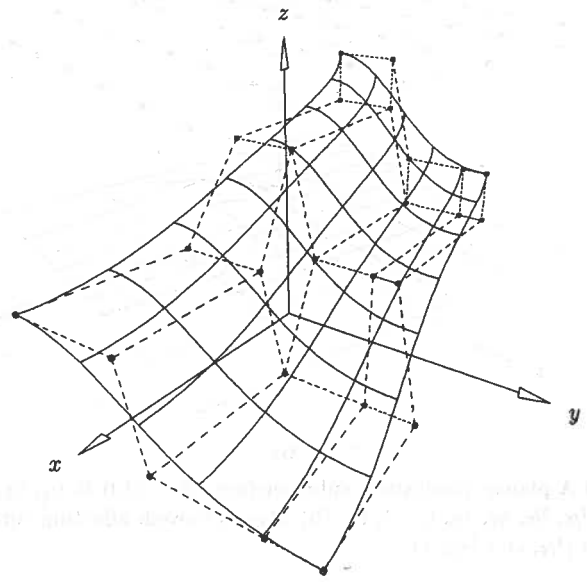
(b)

Figure 3.21. (a) A cubic×quadratic B-spline surface; (b) the strong convex hull property.

P3.23 If triangulated, the control net forms a piecewise planar approximation to the surface; as is the case for curves, the lower the degree the better the approximation (see Figures 3.22a and 3.22b);

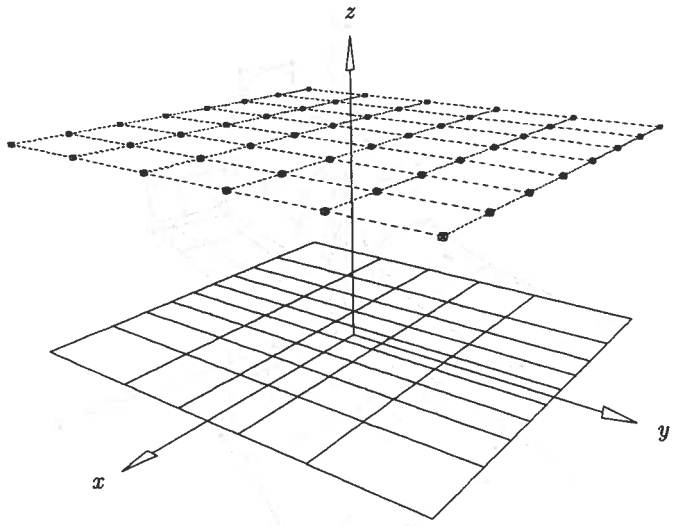


(a)

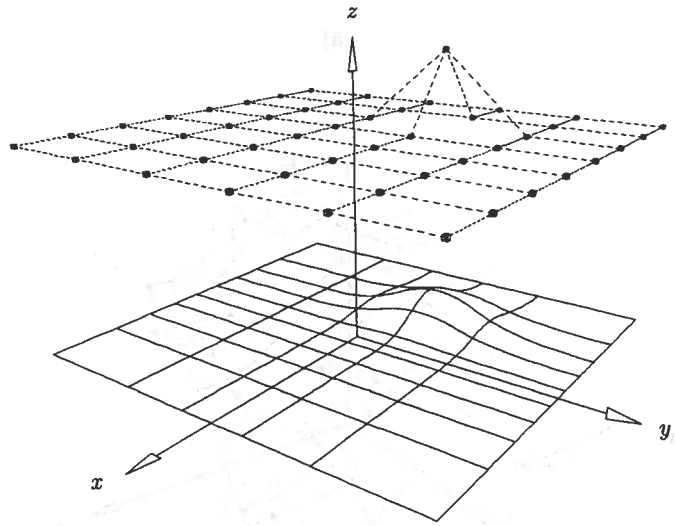


(b)

Figure 3.22. (a) A biquadratic surface; (b) a biquartic surface ($p = q = 4$) using the same control points as in Figure 3.22a.



(a)



(b)

Figure 3.23. (a) A planar quadratic × cubic surface, $U = \{0,0,0,\frac{1}{4},\frac{1}{2},\frac{3}{4},1,1,1\}$ and $V = \{0,0,0,0,\frac{1}{5},\frac{2}{5},\frac{3}{5},\frac{4}{5},1,1,1,1\}$; (b) $\mathbf{P}_{3,5}$ is moved, affecting surface shape only in the rectangle $[\frac{1}{4},1) \times [\frac{2}{5},1)$.

P3.24 Local modification scheme: if $\mathbf{P}_{i,j}$ is moved it affects the surface only in the rectangle $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$; this follows from P3.15. Now consider Figures 3.23a and 3.23b: the initial surface is flat because all the control points lie in a common plane (P3.22); the control net is offset from the surface for better visualization. When $\mathbf{P}_{3,5}$ is moved it affects the surface shape only in the rectangle $[\frac{1}{4},1) \times [\frac{2}{5},1)$;

P3.25 The continuity and differentiability of $\mathbf{S}(u,v)$ follows from that of the basis functions. In particular, $\mathbf{S}(u,v)$ is $p-k$ $(q-k)$ times differentiable in the u (v) direction at a u (v) knot of multiplicity k. Figure 3.24 shows a quadratic × cubic surface defined on the knot vectors $U = \{0,0,0,\frac{1}{2},\frac{1}{2},1,1,1\}$ and $V = \{0,0,0,0,\frac{1}{2},1,1,1,1\}$. Notice the crease in the surface, corresponding to the knot line $u = \frac{1}{2}$. Of course, as is the case for curves, it is possible to position the control points in such a way that they cancel the discontinuities in the basis functions. By using multiply coincident control points, visual discontinuities can be created where there are no corresponding discontinuities in the basis functions; Figure 3.25 shows such a surface, which is bicubic with no multiple knots. Hence, the second partial derivatives are everywhere continuous. The crease is due to the multiple control points.

We remark here that there is no known variation diminishing property for B-spline surfaces (see [Prau92]).

Isoparametric curves on $\mathbf{S}(u,v)$ are obtained in a manner analogous to that for Bézier surfaces. Fix $u = u_0$

$$\mathbf{C}_{u_0}(v) = \mathbf{S}(u_0,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u_0)N_{j,q}(v)\mathbf{P}_{i,j}$$

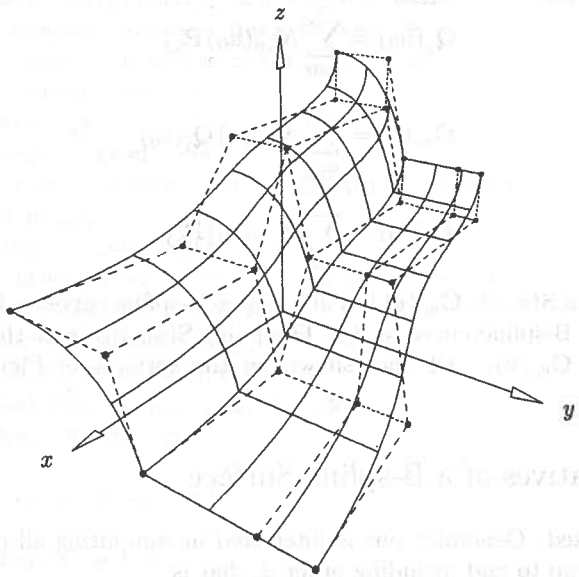


Figure 3.24. A quadratic × cubic surface with crease, $U = \{0,0,0,\frac{1}{2},\frac{1}{2},1,1,1\}$ and $V = \{0,0,0,0,\frac{1}{2},1,1,1,1\}$.

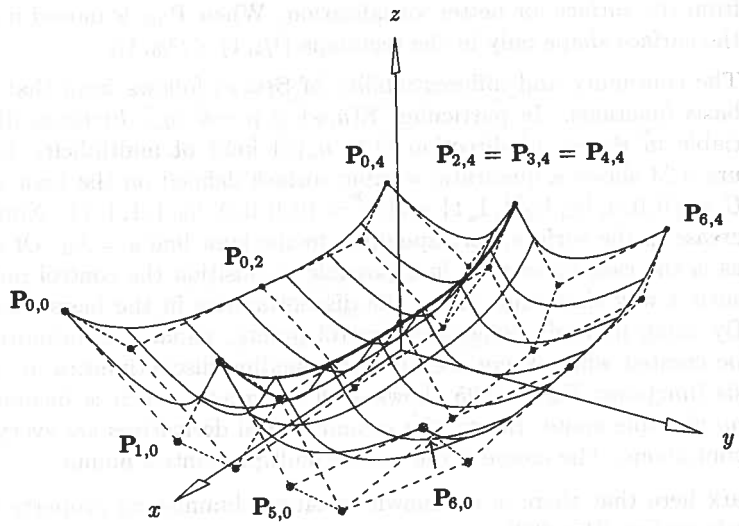


Figure 3.25. A bicubic surface with crease, $U = \{0,0,0,0,\frac{1}{4},\frac{1}{2},\frac{3}{4},1,1,1,1\}$ and $V = \{0,0,0,0,\frac{1}{2},1,1,1,1\}$; $\mathbf{P}_{2,j} = \mathbf{P}_{3,j} = \mathbf{P}_{4,j}$ for $0 \le j \le 4$.

$$= \sum_{j=0}^{m} N_{j,q}(v) \left(\sum_{i=0}^{n} N_{i,p}(u_0) \mathbf{P}_{i,j} \right) = \sum_{j=0}^{m} N_{j,q}(v)\, \mathbf{Q}_j(u_0) \tag{3.14}$$

where
$$\mathbf{Q}_j(u_0) = \sum_{i=0}^{n} N_{i,p}(u_0) \mathbf{P}_{i,j}$$

Analogously
$$\mathbf{C}_{v_0}(u) = \sum_{i=0}^{n} N_{i,p}(u)\, \mathbf{Q}_i(v_0)$$

where
$$\mathbf{Q}_i(v_0) = \sum_{j=0}^{m} N_{j,q}(v_0) \mathbf{P}_{i,j} \tag{3.15}$$

is a u isocurve on $\mathbf{S}(u,v)$. $\mathbf{C}_{u_0}(v)$ is a qth-degree B-spline curve on V, and $\mathbf{C}_{v_0}(u)$ is a pth-degree B-spline curve on U. The point $\mathbf{S}(u_0,v_0)$ is at the intersection of $\mathbf{C}_{u_0}(v)$ and $\mathbf{C}_{v_0}(u)$. All lines shown on the surfaces of Figures 3.20–3.25 are isolines.

3.5 Derivatives of a B-spline Surface

Let (u,v) be fixed. Generally, one is interested in computing all partial derivatives of $\mathbf{S}(u,v)$ up to and including order d, that is

$$\frac{\partial^{k+l}}{\partial^k u\, \partial^l v} \mathbf{S}(u,v) \qquad 0 \le k+l \le d \tag{3.16}$$

As for curves, we obtain these derivatives by computing derivatives of the basis functions (see Eqs. [2.9] and[2.10], and Algorithm A2.3). In particular

$$\frac{\partial^{k+l}}{\partial^k u\, \partial^l v} \mathbf{S}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}^{(k)}\, N_{j,q}^{(l)} \mathbf{P}_{i,j} \tag{3.17}$$

Algorithm A3.6 computes the point on a B-spline surface and all partial derivatives up to and including order d ($d > p, q$ is allowed). Analogous to Algorithm A3.5, this is a five-step process, with the last step being vector/matrix/vector multiplications of the form

$$\frac{\partial^{k+l}}{\partial^k u\, \partial^l v} \mathbf{S}(u,v) = \left[N_{r,p}^{(k)}(u) \right]^T \left[\mathbf{P}_{r,s} \right] \left[N_{s,q}^{(l)}(v) \right]$$
$$0 \le k+l \le d \quad \begin{array}{l} uspan - p \le r \le uspan \\ vspan - q \le s \le vspan \end{array} \tag{3.18}$$

Output is the array `SKL[][]`, where `SKL[k][l]` is the derivative of $\mathbf{S}(u,v)$ with respect to u k times, and v l times. For fixed k, $0 \le k \le d$, local array `temp[]` stores the vector/matrix product, $\left[N_{r,p}{}^{(k)}(u) \right]^T \left[\mathbf{P}_{r,s} \right]$, while it is being multiplied with the $\left[N_{s,q}{}^{(l)}(v) \right]$, for $0 \le l \le d-k$. Arrays `Nu[][]` and `Nv[][]` are used to store the derivatives of the basis functions.

```
ALGORITHM A3.6
  SurfaceDerivsAlg1(n,p,U,m,q,V,P,u,v,d,SKL)
    { /*  Compute surface derivatives  */
      /*  Input:  n,p,U,m,q,V,P,u,v,d  */
      /*  Output: SKL  */
    du = min(d,p);
    for (k=p+1; k<=d; k++)
      for (l=0; l<=d-k; l++)    SKL[k][l] = 0.0;
    dv = min(d,q);
    for (l=q+1; l<=d; l++)
      for (k=0; k<=d-l; k++)    SKL[k][l] = 0.0;
    uspan = FindSpan(n,p,u,U);
    DersBasisFuns(uspan,u,p,du,U,Nu);
    vspan = FindSpan(m,q,v,V);
    DersBasisFuns(vspan,v,q,dv,V,Nv);
    for (k=0; k<=du; k++)
      {
      for (s=0; s<=q; s++)
        {
        temp[s] = 0.0;
        for (r=0; r<=p; r++)
          temp[s] = temp[s] + Nu[k][r]*P[uspan-p+r][vspan-q+s];
        }
```

```
        dd = min(d-k,dv);
        for (l=0; l<=dd; l++)
          {
          SKL[k][l] = 0.0;
          for (s=0; s<=q; s++)
            SKL[k][l] = SKL[k][l] + Nv[l][s]*temp[s];
          }
      }
    }
```

Figure 3.26 shows a bicubic surface and its first and second partial derivatives. Note that the derivatives are scaled down by $1/2$ for better visualization.

Let us formally differentiate $\mathbf{S}(u,v)$. With respect to u we have

$$\mathbf{S}_u(u,v) = \frac{\partial}{\partial u}\mathbf{S}(u,v) = \sum_{j=0}^{m} N_{j,q}(v)\left(\frac{\partial}{\partial u}\sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_{i,j}\right) \tag{3.19}$$

$$= \sum_{j=0}^{m} N_{j,q}(v)\left(\frac{\partial}{\partial u}\mathbf{C}_j(u)\right)$$

where $$\mathbf{C}_j(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_{i,j} \qquad j = 0,\ldots,m$$

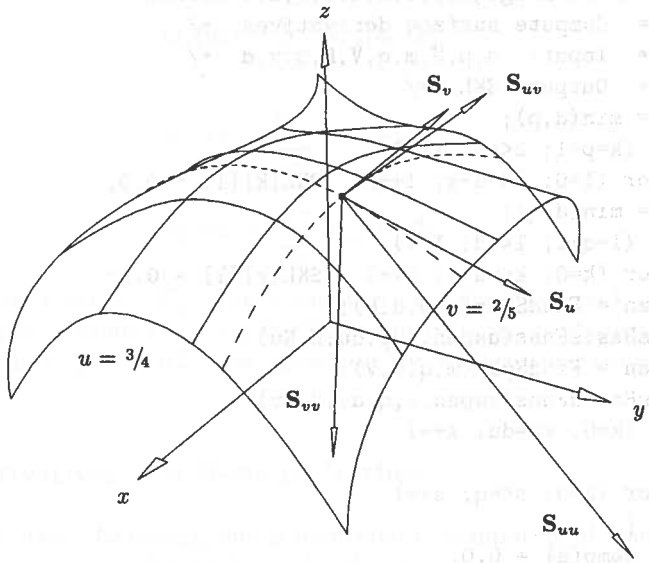


Figure 3.26. A bicubic surface defined on $U = V = \{0,0,0,0,1/2,1,1,1,1\}$ and its first and second partial derivatives computed at $u = 3/4$ and $v = 2/5$.

are B-spline curves. Applying Eq. (3.6) to each of the $\mathbf{C}_j(u)$ and substituting into Eq. (3.19), we obtain

$$\mathbf{S}_u(u,v) = \sum_{i=0}^{n-1}\sum_{j=0}^{m} N_{i,p-1}(u)N_{j,q}(v)\mathbf{P}_{i,j}^{(1,0)} \tag{3.20}$$

where $$\mathbf{P}_{i,j}^{(1,0)} = p\frac{\mathbf{P}_{i+1,j} - \mathbf{P}_{i,j}}{u_{i+p+1} - u_{i+1}} \quad \text{(see Eq. [3.4])}$$

$$U^{(1)} = \{\underbrace{0,\ldots,0}_{p}, u_{p+1},\ldots,u_{r-p-1}, \underbrace{1,\ldots,1}_{p}\}$$

$$V^{(0)} = V$$

Analogously $$\mathbf{S}_v(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m-1} N_{i,p}(u)N_{j,q-1}(v)\mathbf{P}_{i,j}^{(0,1)} \tag{3.21}$$

where $$\mathbf{P}_{i,j}^{(0,1)} = q\frac{\mathbf{P}_{i,j+1} - \mathbf{P}_{i,j}}{v_{j+q+1} - v_{j+1}}$$

$$U^{(0)} = U$$

$$V^{(1)} = \{\underbrace{0,\ldots,0}_{q}, v_{q+1},\ldots,v_{s-q-1}, \underbrace{1,\ldots,1}_{q}\}$$

Applying first Eq. (3.20), then Eq. (3.21) yields

$$\mathbf{S}_{uv}(u,v) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} N_{i,p-1}(u)N_{j,q-1}(v)\mathbf{P}_{i,j}^{(1,1)} \tag{3.22}$$

where $$\mathbf{P}_{i,j}^{(1,1)} = q\frac{\mathbf{P}_{i,j+1}^{(1,0)} - \mathbf{P}_{i,j}^{(1,0)}}{v_{j+q+1} - v_{j+1}}$$

and $U^{(1)}$ and $V^{(1)}$ are as defined previously.

In general

$$\frac{\partial^{k+l}}{\partial^k u\,\partial^l v}\mathbf{S}(u,v) = \sum_{i=0}^{n-k}\sum_{j=0}^{m-l} N_{i,p-k}(u)N_{j,q-l}(v)\mathbf{P}_{i,j}^{(k,l)} \tag{3.23}$$

where $$\mathbf{P}_{i,j}^{(k,l)} = (q-l+1)\frac{\mathbf{P}_{i,j+1}^{(k,l-1)} - \mathbf{P}_{i,j}^{(k,l-1)}}{v_{j+q+1} - v_{j+l}}$$

Using Eqs. (3.20)–(3.23), we derive useful formulas for corner derivatives. For example, at the corner $(u,v) = (0,0)$, we have

$$\begin{aligned}
\mathbf{S}_u(0,0) &= \mathbf{P}_{0,0}^{(1,0)} = \frac{p}{u_{p+1}}(\mathbf{P}_{1,0} - \mathbf{P}_{0,0}) \\
\mathbf{S}_v(0,0) &= \mathbf{P}_{0,0}^{(0,1)} = \frac{q}{v_{q+1}}(\mathbf{P}_{0,1} - \mathbf{P}_{0,0}) \\
\mathbf{S}_{uv}(0,0) &= \mathbf{P}_{0,0}^{(1,1)} = \frac{q}{v_{q+1}}\left(\mathbf{P}_{0,1}^{(1,0)} - \mathbf{P}_{0,0}^{(1,0)}\right) \\
&= \frac{pq}{u_{p+1}v_{q+1}}(\mathbf{P}_{1,1} - \mathbf{P}_{0,1} - \mathbf{P}_{1,0} + \mathbf{P}_{0,0})
\end{aligned} \tag{3.24}$$

Now let $u_0 = 0$ and $v_0 = 0$. From the properties of the basis functions, it is easy to see that the isocurves $\mathbf{C}_{u_0}(v)$ and $\mathbf{C}_{v_0}(u)$ are given by

$$\mathbf{C}_{u_0}(v) = \sum_{j=0}^{m} N_{j,q}(v)\mathbf{P}_{0,j} \qquad \mathbf{C}_{v_0}(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_{i,0}$$

From Eq. (3.7) it follows that

$$\mathbf{S}_u(0,0) = \mathbf{C}'_{v_0}(0) \qquad \mathbf{S}_v(0,0) = \mathbf{C}'_{u_0}(0)$$

Algorithm A3.7 computes all (or optionally some) of the control points, $\mathbf{P}_{i,j}^{(k,l)}$, of the derivative surfaces up to order d ($0 \le k+l \le d$). The algorithm is based on Eq. (3.23) and Algorithm A3.3. Output is the array, PKL[][][][], where PKL[k][l][i][j] is the i,jth control point of the surface, differentiated k times with respect to u and l times with respect to v.

```
ALGORITHM A3.7
  SurfaceDerivCpts(n,p,U,m,q,V,P,d,r1,r2,s1,s2,PKL)
    { /*  Compute control points of derivative surfaces  */
      /*  Input:  n,p,U,m,q,V,P,d,r1,r2,s1,s2  */
      /*  Output: PKL  */
    du = min(d,p);     dv = min(d,q);
    r = r2-r1;     s = s2-s1;
    for (j=s1; j<=s2; j++)
      {
      CurveDerivCpts(n,p,U,&P[][j],du,r1,r2,temp);
      for (k=0; k<=du; k++)
        for (i=0; i<=r-k; i++)
          PKL[k][0][i][j-s1] = temp[k][i];
      }
    for (k=0; k<du; k++)
      for (i=0; i<=r-k; i++)
        {
```



```
        dd = min(d-k,dv);
        CurveDerivCpts(m,q,&V[s1],&PKL[k][0][i][],dd,0,s,temp);
        for (l=1; l<=dd; l++)
          for (j=0; j<=s-l; j++)
            PKL[k][l][i][j] = temp[l][j];
        }
    }
```

Algorithm A3.8 computes the point on a B-spline surface and all partial derivatives up to and including order d, at fixed parameters (u,v) (compare with Algorithm A3.6). $d > p,q$ is allowed. On output, SKL[k][l] is the derivative of $\mathbf{S}(u,v)$ k times with respect to u and l times with respect to v.

```
ALGORITHM A3.8:
  SurfaceDerivsAlg2(n,p,U,m,q,V,P,u,v,d,SKL)
    { /*  Compute surface derivatives  */
      /*  Input:  n,p,U,m,q,V,P,u,v,d  */
      /*  Output: SKL  */
    du = min(d,p);
    for (k=p+1; k<=d; k++)
      for (l=0; l<=d-k; l++)    SKL[k][l] = 0.0;
    dv = min(d,q);
    for (l=q+1; l<=d; l++)
      for (k=0; k<=d-l; k++)    SKL[k][l] = 0.0;
    uspan = FindSpan(n,p,u,U);
    AllBasisFuns(uspan,u,p,U,Nu);
    vspan = FindSpan(m,q,v,V);
    AllBasisFuns(vspan,v,q,V,Nv);
    SurfaceDerivCpts(n,p,U,m,q,V,P,d,uspan-p,uspan,
                                              vspan-q,vspan,PKL);
    for (k=0; k<=du; k++)
      {
      dd = min(d-k,dv);
      for (l=0; l<=dd; l++)
        {
        SKL[k][l] = 0.0;
        for (i=0; i<=q-l; i++)
          {
          tmp = 0.0;
          for (j=0; j<=p-k; j++)
            tmp = tmp + Nu[j][p-k]*PKL[k][l][j][i];
          SKL[k][l] = SKL[k][l] + Nv[i][q-l]*tmp;
          }
        }
      }
    }
```

EXERCISES

3.1. Why do quadratic curves touch their control polygons at knots?

3.2. If a quadratic curve has an inflection point, it must be at a knot (see the figures). Why?

3.3. Construct a C^2 continuous cubic curve with a cusp.

3.4. Let a cubic curve be defined by $\mathbf{C}(u) = \sum_{i=0}^{7} N_{i,3}(u)\mathbf{P}_i$ and the knot vector $U = \{0,0,0,0,\frac{1}{4},\frac{1}{4},\frac{2}{3},\frac{3}{4},1,1,1,1\}$.

a. Assume some arbitrary locations for the $\mathbf{P}_i$ and sketch the curve.

b. Where is the point $\mathbf{C}(\frac{1}{4})$?

c. If $\mathbf{P}_2$ is moved, on what subinterval of $[0,1]$ is $\mathbf{C}(u)$ affected? If $\mathbf{P}_5$ is moved, what subinterval is affected?

d. Which control points are affecting curve shape on the interval $u \in [\frac{1}{4},\frac{2}{3})$? On the interval $u \in [\frac{2}{3},\frac{3}{4})$?

3.5. Let $\mathbf{C}(u) = \sum_{i=0}^{3} N_{i,2}(u)\mathbf{P}_i$, where $U = \{0,0,0,\frac{1}{2},1,1,1\}$ and $\mathbf{P}_0 = (-1,0)$, $\mathbf{P}_1 = (-1,1)$, $\mathbf{P}_2 = (1,1)$, and $\mathbf{P}_3 = (1,0)$. Sketch $\mathbf{C}(u)$. Compute $\mathbf{C}'(u)$, i.e., its control points and knot vector. Sketch $\mathbf{C}'(u)$.

3.6. For the cubic curve of Figure 3.16a, assume the control points $\{\mathbf{P}_i\} = \{(0,0), (1,2), (3,4), (5,2), (5,0), (8,0), (9,3)\}$. Sketch the curve. Using Eq. (3.8), compute the second derivative curve, $\mathbf{C}^{(2)}(u)$. Sketch $\mathbf{C}^{(2)}(u)$. Let $\mathbf{P}_0^{(2)}$ and $\mathbf{P}_4^{(2)}$ be the first and last control points of $\mathbf{C}^{(2)}(u)$, i.e., $\mathbf{P}_0^{(2)} = \mathbf{C}^{(2)}(0)$ and $\mathbf{P}_4^{(2)} = \mathbf{C}^{(2)}(1)$. Compute $\mathbf{C}^{(2)}(0)$ and $\mathbf{C}^{(2)}(1)$ using Eqs. (3.9) and (3.10).

3.7. A crease in a surface can be created using either multiple knots or multiple control points (see Figures 3.24 and 3.25). Which would you use if you wanted a crease running less than the full length of the surface? Construct and sketch such an example.

3.8. 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
$$U = \{0,0,0,\tfrac{1}{2},1,1,1\}$$
$$V = \{0,0,0,1,1,1\}$$

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 $\mathbf{S}(\frac{3}{10},\frac{6}{10})$ by evaluating the nonzero B-spline basis functions and multiplying these by the appropriate control points.

3.9. Derive the expressions for $\mathbf{S}_u(u,v)$, $\mathbf{S}_v(u,v)$, and $\mathbf{S}_{uv}(u,v)$ at the three corners, $(u,v) = (0,1)$, $(1,0)$, and $(1,1)$ (see Eq. [3.24]).

3.10. Let $\mathbf{S}(u,v)$ be as in Exercise 3.8. Sketch this surface. Using Eqs. (3.20) and (3.21), compute the surfaces $\mathbf{S}_u(u,v)$ and $\mathbf{S}_v(u,v)$. Sketch these two surfaces. Using Eq. (3.22) and the expressions derived in Exercise 3.9, compute the mixed partial derivative, $\mathbf{S}_{uv}(u,v)$, at each of the four corners of the surface. What is the geometric significance of these four vectors?

CHAPTER

FOUR

Rational B-spline Curves and Surfaces

4.1 Introduction

In this chapter we combine the concepts of Sections 1.4 and 1.5 of Chapter 1 and those of Chapter 3 to obtain **N**on**U**niform **R**ational **B**-**S**pline (NURBS) curves and surfaces. We present definitions and general properties and derive formulas and algorithms for the derivatives of NURBS curves and surfaces in terms of their nonrational counterparts. The earliest published works on NURBS are [Vers75; Till83]. A more recent survey can be found in [Pieg91a].

4.2 Definition and Properties of NURBS Curves

A pth-degree NURBS curve is defined by

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u)w_i\mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u)w_i} \qquad a \le u \le b \tag{4.1}$$

where the $\{\mathbf{P}_i\}$ are the *control points* (forming a *control polygon*), the $\{w_i\}$ are the *weights*, and the $\{N_{i,p}(u)\}$ are the pth-degree B-spline basis functions defined on the nonperiodic (and nonuniform) knot vector

$$U = \{\underbrace{a,\ldots,a}_{p+1}, u_{p+1},\ldots,u_{m-p-1}, \underbrace{b,\ldots,b}_{p+1}\}$$

Unless otherwise stated, we assume that $a = 0$, $b = 1$, and $w_i > 0$ for all i. Setting

$$R_{i,p}(u) = \frac{N_{i,p}(u)w_i}{\sum_{j=0}^{n} N_{j,p}(u)w_j} \tag{4.2}$$

allows us to rewrite Eq. (4.1) in the form

$$\mathbf{C}(u) = \sum_{i=0}^{n} R_{i,p}(u)\,\mathbf{P}_i \tag{4.3}$$

The $\{R_{i,p}(u)\}$ are the *rational basis functions*; they are piecewise rational functions on $u \in [0,1]$.

The $R_{i,p}(u)$ have the following properties derived from Eq. (4.2) and the corresponding properties of the $N_{i,p}(u)$:

P4.1 Nonnegativity: $R_{i,p}(u) \geq 0$ for all i, p, and $u \in [0,1]$;

P4.2 Partition of unity: $\sum_{i=0}^{n} R_{i,p}(u) = 1$ for all $u \in [0,1]$;

P4.3 $R_{0,p}(0) = R_{n,p}(1) = 1$;

P4.4 For $p > 0$, all $R_{i,p}(u)$ attain exactly one maximum on the interval $u \in [0,1]$;

P4.5 Local support: $R_{i,p}(u) = 0$ for $u \notin [u_i, u_{i+p+1})$. Furthermore, in any given knot span, at most $p+1$ of the $R_{i,p}(u)$ are nonzero (in general, $R_{i-p,p}(u), \ldots, R_{i,p}(u)$ are nonzero in $[u_i, u_{i+1})$);

P4.6 All derivatives of $R_{i,p}(u)$ exist in the interior of a knot span, where it is a rational function with nonzero denominator. At a knot, $R_{i,p}(u)$ is $p-k$ times continuously differentiable, where k is the multiplicity of the knot;

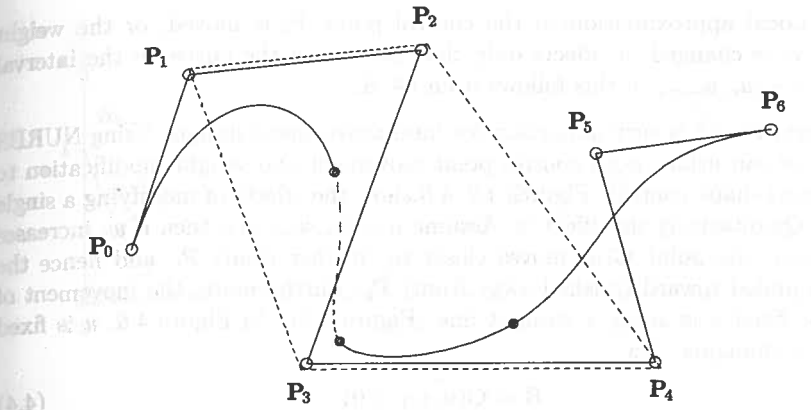
P4.7 If $w_i = 1$ for all i, then $R_{i,p}(u) = N_{i,p}(u)$ for all i; i.e., the $N_{i,p}(u)$ are special cases of the $R_{i,p}(u)$. In fact, for any $a \neq 0$, if $w_i = a$ for all i then $R_{i,p}(u) = N_{i,p}(u)$ for all i.

Properties P4.1–P4.7 yield the following important geometric characteristics of NURBS curves:

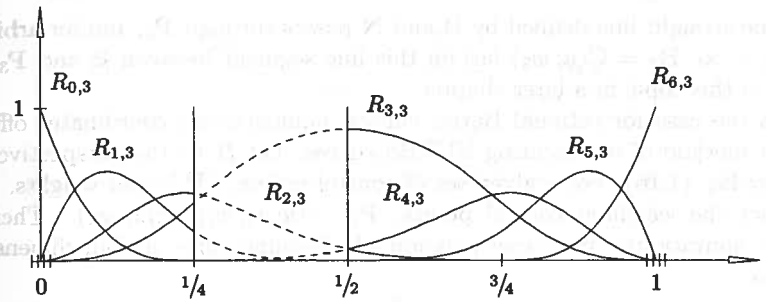
P4.8 $\mathbf{C}(0) = \mathbf{P}_0$ and $\mathbf{C}(1) = \mathbf{P}_n$; this follows from P4.3;

P4.9 Affine invariance: an affine transformation is applied to the curve by applying it to the control points (see P3.4, Section 3.1); NURBS curves are also invariant under perspective projections ([Lee87; Pieg91a]), a fact which is important in computer graphics;

P4.10 Strong convex hull property: if $u \in [u_i, u_{i+1})$, then $\mathbf{C}(u)$ lies within the convex hull of the control points $\mathbf{P}_{i-p}, \ldots, \mathbf{P}_i$ (see Figure 4.1, where $\mathbf{C}(u)$ for $u \in [1/4, 1/2)$ (dashed segment) is contained in the convex hull of $\{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4\}$, the dashed area); this follows from P4.1, P4.2, and P4.5;



(a)



(b)

Figure 4.1. $U = \{0,0,0,0,1/4,1/2,3/4,1,1,1,1\}$ and $\{w_0, \ldots, w_6\} = \{1,1,1,3,1,1,1\}$. (a) A cubic NURBS curve; (b) associated basis functions.

P4.11 $\mathbf{C}(u)$ is infinitely differentiable on the interior of knot spans and is $p-k$ times differentiable at a knot of multiplicity k;

P4.12 Variation diminishing property: no plane has more intersections with the curve than with the control polygon (replace the word 'plane', with 'line' for two-dimensional curves);

P4.13 A NURBS curve with no interior knots is a rational Bézier curve, since the $N_{i,p}(u)$ reduce to the $B_{i,n}(u)$; compare Eqs. (4.2) and (4.3) with Eq. (1.15). This, together with P4.7, implies that NURBS curves contain nonrational B-spline and rational and nonrational Bézier curves as special cases;

P4.14 Local approximation: if the control point $\mathbf{P}_i$ is moved, or the weight w_i is changed, it affects only that portion of the curve on the interval $u \in [u_i, u_{i+p+1})$; this follows from P4.5.

Property P4.14 is very important for interactive shape design. Using NURBS curves, we can utilize both control point movement and weight modification to attain local shape control. Figures 4.2–4.6 show the effects of modifying a single weight. Qualitatively the effect is: Assume $u \in [u_i, u_{i+p+1})$; then if w_i increases (decreases), the point $\mathbf{C}(u)$ moves closer to (farther from) $\mathbf{P}_i$, and hence the curve is pulled toward (pushed away from) $\mathbf{P}_i$. Furthermore, the movement of $\mathbf{C}(u)$ for fixed u is along a straight line (Figure 4.6). In Figure 4.6, u is fixed and w_3 is changing. Let

$$\mathbf{B} = \mathbf{C}(u; w_3 = 0) \tag{4.4}$$

$$\mathbf{N} = \mathbf{C}(u; w_3 = 1)$$

Then the straight line defined by $\mathbf{B}$ and $\mathbf{N}$ passes through $\mathbf{P}_3$, and for arbitrary $0 < w_3 < \infty$, $\mathbf{B}_3 = \mathbf{C}(u; w_3)$ lies on this line segment between $\mathbf{B}$ and $\mathbf{P}_3$. We return to this topic in a later chapter.

As is the case for rational Bézier curves, homogeneous coordinates offer an efficient method of representing NURBS curves. Let H be the perspective map given by Eq. (1.16). For a given set of control points, $\{\mathbf{P}_i\}$, and weights, $\{w_i\}$, construct the weighted control points, $\mathbf{P}_i^w = (w_i x_i, w_i y_i, w_i z_i, w_i)$. Then define the nonrational (piecewise polynomial) B-spline curve in four-dimensional space as

$$\mathbf{C}^w(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i^w \tag{4.5}$$

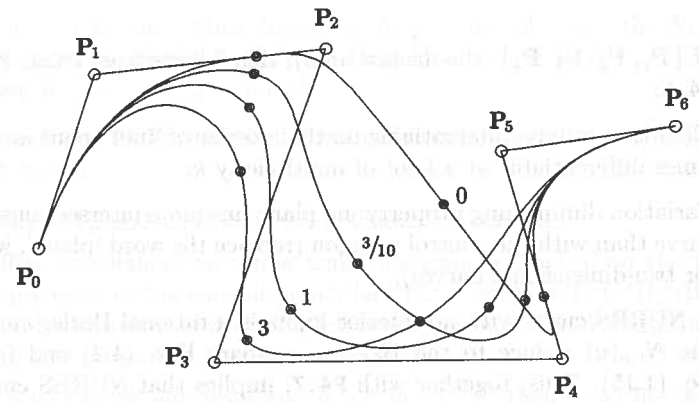
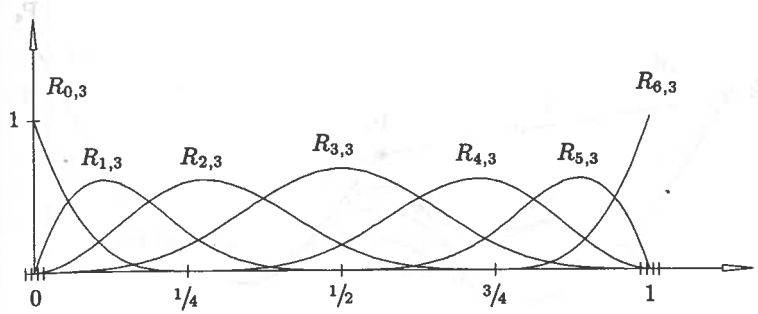
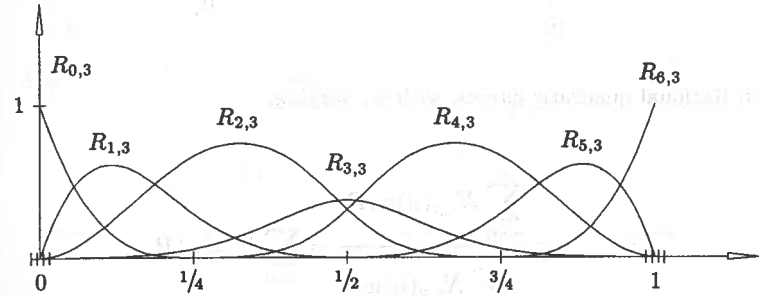


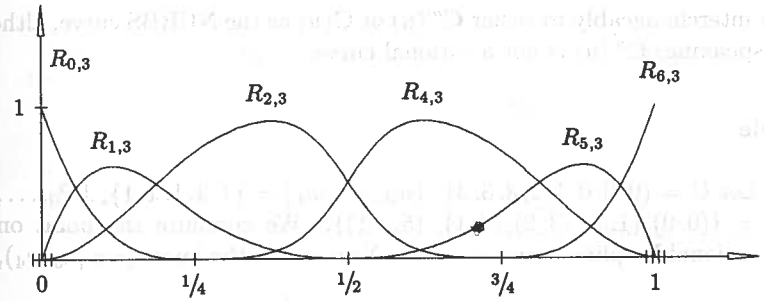
Figure 4.2. Rational cubic B-spline curves, with w_3 varying.



(a)



(b)



(c)

Figure 4.3. The cubic basis functions for the curves of Figure 4.2. (a) $w_3 = 1$; (b) $w_3 = 3/10$; (c) $w_3 = 0$.

Applying the perspective map, H, to $\mathbf{C}^w(u)$ yields the corresponding rational B-spline curve (piecewise rational in three-dimensional space)

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

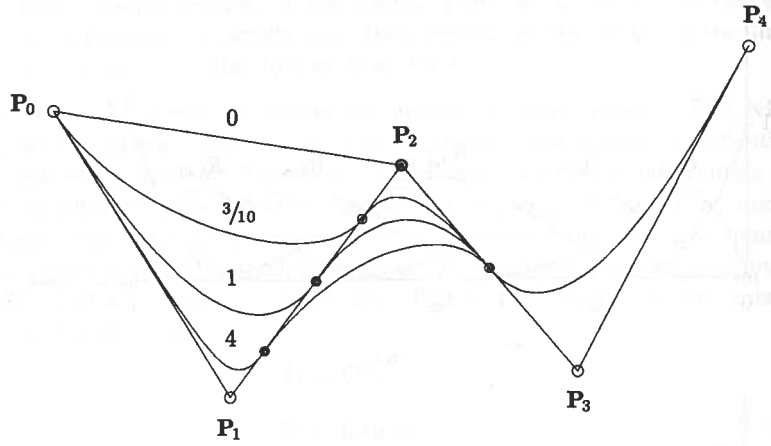


Figure 4.4. Rational quadratic curves, with w_1 varying.

$$= \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} = \sum_{i=0}^{n} R_{i,p}(u) \mathbf{P}_i$$

We refer interchangeably to either $\mathbf{C}^w(u)$ or $\mathbf{C}(u)$ as the NURBS curve, although strictly speaking, $\mathbf{C}^w(u)$ is not a rational curve.

Example

Ex4.1 Let $U = \{0,0,0,1,2,3,3,3\}$, $\{w_0, \ldots, w_4\} = \{1,4,1,1,1\}$, $\{\mathbf{P}_0, \ldots, \mathbf{P}_4\} = \{(0,0),(1,1),(3,2),(4,1),(5,-1)\}$. We compute the point on the rational B-spline curve at $u = 1$. Now u is in the knot span $[u_3, u_4)$, and

$$N_{3,0}(1) = 1$$

$$N_{2,1}(1) = \frac{2-1}{2-1} N_{3,0}(1) = 1$$

$$N_{3,1}(1) = \frac{1-1}{2-1} N_{3,0}(1) = 0$$

$$N_{1,2}(1) = \frac{2-1}{2-0} N_{2,1}(1) = \frac{1}{2}$$

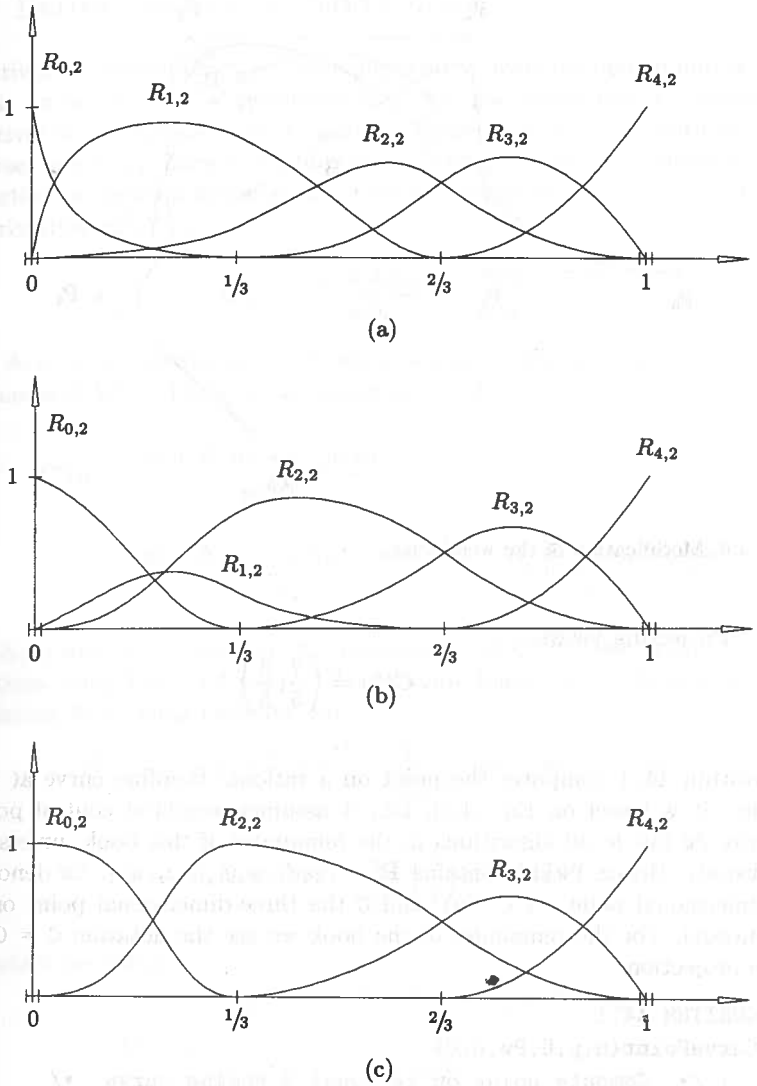


Figure 4.5. The quadratic basis functions for the curves of Figure 4.4. (a) $w_1 = 4$; (b) $w_1 = 3/10$; (c) $w_1 = 0$.

$$N_{2,2}(1) = \frac{1-0}{2-0} N_{2,1}(1) = \frac{1}{2}$$

$$N_{3,2}(1) = 0$$

Hence

$$\mathbf{C}^w(1) = \frac{1}{2}\mathbf{P}_1^w + \frac{1}{2}\mathbf{P}_2^w = \frac{1}{2}(4,4,4) + \frac{1}{2}(3,2,1) = \left(\frac{7}{2}, 3, \frac{5}{2}\right)$$

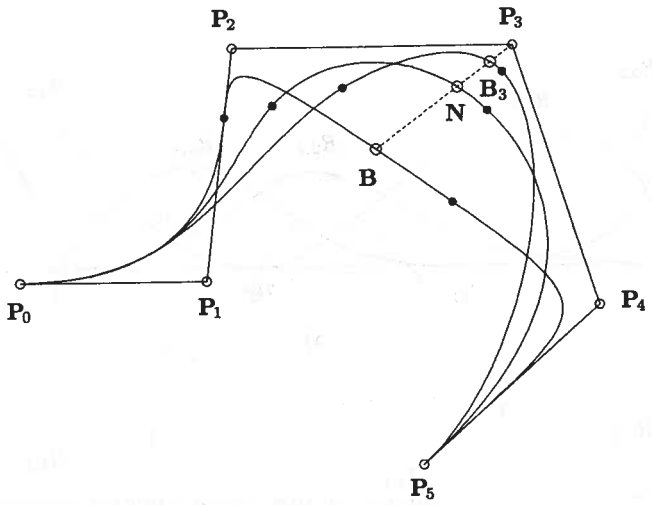


Figure 4.6. Modification of the weight w_3.

Projecting yields

$$\mathbf{C}(1) = \left(\frac{7}{5}, \frac{6}{5}\right)$$

Algorithm A4.1 computes the point on a rational B-spline curve at a fixed u value. It is based on Eq. (4.5), i.e., it assumes weighted control points in the array Pw (as do all algorithms in the remainder of this book, unless otherwise stated). Hence, Pw[i] contains $\mathbf{P}_i = (w_i x_i, w_i y_i, w_i z_i, w_i)$. Cw denotes the four-dimensional point on $\mathbf{C}^w(u)$, and C the three-dimensional point on $\mathbf{C}(u)$ (the output). For the remainder of the book we use the notation C = Cw/w to denote projection.

```
ALGORITHM A4.1
  CurvePoint(n,p,U,Pw,u,C)
    { /*  Compute point on rational B-spline curve  */
      /*  Input:  n,p,U,Pw,u  */
      /*  Output: C  */
    span = FindSpan(n,p,u,U);
    BasisFuns(span,u,p,U,N);
    Cw = 0.0;
    for (j=0; j<=p; j++)
      Cw = Cw + N[j]*Pw[span-p+j];
    C = Cw/w;    /*  Divide by weight  */
    }
```


4.3 Derivatives of a NURBS Curve

Derivatives of rational functions are complicated, involving denominators to high powers. In Section 3.3 we developed formulas and algorithms to compute the derivatives of nonrational B-spline curves. Those formulas and algorithms apply, of course, to $\mathbf{C}^w(u)$, since it is a nonrational curve in four-dimensional space. In this section we develop formulas that express the derivatives of $\mathbf{C}(u)$ in terms of the derivatives of $\mathbf{C}^w(u)$.

Let

$$\mathbf{C}(u) = \frac{w(u)\mathbf{C}(u)}{w(u)} = \frac{\mathbf{A}(u)}{w(u)}$$

where $\mathbf{A}(u)$ is the vector-valued function whose coordinates are the first three coordinates of $\mathbf{C}^w(u)$ ($\mathbf{A}(u)$ is the numerator of Eq. [4.1]). Then

$$\mathbf{C}'(u) = \frac{w(u)\mathbf{A}'(u) - w'(u)\mathbf{A}(u)}{w(u)^2}$$

$$= \frac{w(u)\mathbf{A}'(u) - w'(u)w(u)\mathbf{C}(u)}{w(u)^2} = \frac{\mathbf{A}'(u) - w'(u)\mathbf{C}(u)}{w(u)} \tag{4.7}$$

Since $\mathbf{A}(u)$ and $w(u)$ represent the coordinates of $\mathbf{C}^w(u)$, we obtain their first derivatives using Eqs. (3.4)–(3.6). We compute higher order derivatives by differentiating $\mathbf{A}(u)$ using Leibnitz' rule

$$\mathbf{A}^{(k)}(u) = \left(w(u)\mathbf{C}(u)\right)^{(k)} = \sum_{i=0}^{k} \binom{k}{i} w^{(i)}(u)\mathbf{C}^{(k-i)}(u)$$

$$= w(u)\mathbf{C}^{(k)}(u) + \sum_{i=1}^{k} \binom{k}{i} w^{(i)}(u)\mathbf{C}^{(k-i)}(u)$$

from which we obtain

$$\mathbf{C}^{(k)}(u) = \frac{\mathbf{A}^{(k)}(u) - \sum_{i=1}^{k} \binom{k}{i} w^{(i)}(u)\mathbf{C}^{(k-i)}(u)}{w(u)} \tag{4.8}$$

Equation (4.8) gives the kth derivative of $\mathbf{C}(u)$ in terms of the kth derivative of $\mathbf{A}(u)$, and the first through $(k-1)$th derivatives of $\mathbf{C}(u)$ and $w(u)$. The derivatives $\mathbf{A}^{(k)}(u)$ and $w^{(i)}(u)$ are obtained using either Eq. (3.3) and Algorithm A3.2 or Eq. (3.8) and Algorithm A3.4.

Let us derive expressions for the first derivatives of a NURBS curve at its endpoints ($u = 0, u = 1$). From Eq. (3.7) we have

$$\mathbf{A}'(0) = \frac{p}{u_{p+1}}(w_1\mathbf{P}_1 - w_0\mathbf{P}_0) \qquad w'(0) = \frac{p}{u_{p+1}}(w_1 - w_0)$$

and from Eq. (4.7)

$$\mathbf{C}'(0) = \frac{\frac{p}{u_{p+1}}(w_1\mathbf{P}_1 - w_0\mathbf{P}_0) - \frac{p}{u_{p+1}}(w_1 - w_0)\mathbf{P}_0}{w_0}$$

from which follows

$$\mathbf{C}'(0) = \frac{p}{u_{p+1}}\frac{w_1}{w_0}(\mathbf{P}_1 - \mathbf{P}_0) \tag{4.9}$$

Analogously

$$\mathbf{C}'(1) = \frac{p}{1 - u_{m-p-1}}\frac{w_{n-1}}{w_n}(\mathbf{P}_n - \mathbf{P}_{n-1}) \tag{4.10}$$

Example

Ex4.2 Consider the quadratic rational Bézier circular arc given in Section 1.4 (Figure 1.19b). This is a NURBS curve on the knot vector $U = \{0,0,0,1,1,1\}$, with $\{\mathbf{P}_i\} = \{(1,0),(1,1),(0,1)\}$ and $\{w_i\} = \{1,1,2\}$. From Eqs. (4.9) and (4.10) we have

$$\mathbf{C}'(0) = \frac{2}{1}\frac{1}{1}(\mathbf{P}_1 - \mathbf{P}_0) = (0,2)$$

$$\mathbf{C}'(1) = \frac{2}{1-0}\frac{1}{2}(\mathbf{P}_2 - \mathbf{P}_1) = (-1,0)$$

From Eq. (4.8)

$$\mathbf{C}''(0) = \frac{\mathbf{A}''(0) - 2w'(0)\mathbf{C}'(0) - w''(0)\mathbf{C}(0)}{w_0}$$

From Eq. (3.9) or Eq. (1.10)

$$\mathbf{A}''(0) = 2(w_0\mathbf{P}_0 - 2w_1\mathbf{P}_1 + w_2\mathbf{P}_2)$$

and

$$w''(0) = 2(w_0 - 2w_1 + w_2)$$

From $w'(0) = 2(w_1 - w_0)$ it follows that

$$\mathbf{C}''(0) = \frac{2}{w_0}[w_0\mathbf{P}_0 - 2w_1\mathbf{P}_1 + w_2\mathbf{P}_2 - 4(w_1 - w_0)(\mathbf{P}_1 - \mathbf{P}_0)$$

$$-(w_0 - 2w_1 + w_2)\mathbf{P}_0]$$

$$= 2(\mathbf{P}_0 - 2\mathbf{P}_1 + 2\mathbf{P}_2 - \mathbf{P}_0) = 4(\mathbf{P}_2 - \mathbf{P}_1) = (-4,0)$$

The computation of $\mathbf{C}''(1)$ is left as an exercise.

Now assume that u is fixed, and that the zeroth through the dth derivatives of $\mathbf{A}(u)$ and $w(u)$ have been computed and loaded into the arrays `Aders` and `wders`, respectively, i.e., $\mathbf{C}^w(u)$ has been differentiated and its coordinates separated off into `Aders` and `wders`. Algorithm `A4.2` computes the point, $\mathbf{C}(u)$, and the derivatives, $\mathbf{C}^{(k)}(u)$, $1 \le k \le d$. The curve point is returned in `CK[0]` and the kth derivative is returned in `CK[k]`. The array `Bin[][]` contains the precomputed binomial coefficients (`Bin[k][i]` is $\binom{k}{i}$).

```
ALGORITHM A4.2
  RatCurveDerivs(Aders,wders,d,CK)
    { /*  Compute C(u) derivatives from Cw(u) derivatives  */
      /*  Input:  Aders,wders,d  */
      /*  Output: CK  */
    for (k=0; k<=d; k++)
      {
      v = Aders[k];
      for (i=1; i<=k; i++)
        v = v - Bin[k][i]*wders[i]*CK[k-i];
      CK[k] = v/wders[0];
      }
    }
```

Figure 4.7 shows the first, second, and third derivatives of a cubic NURBS curve. The derivative vectors are scaled down by 0.4, 0.08, and 0.03, respectively.

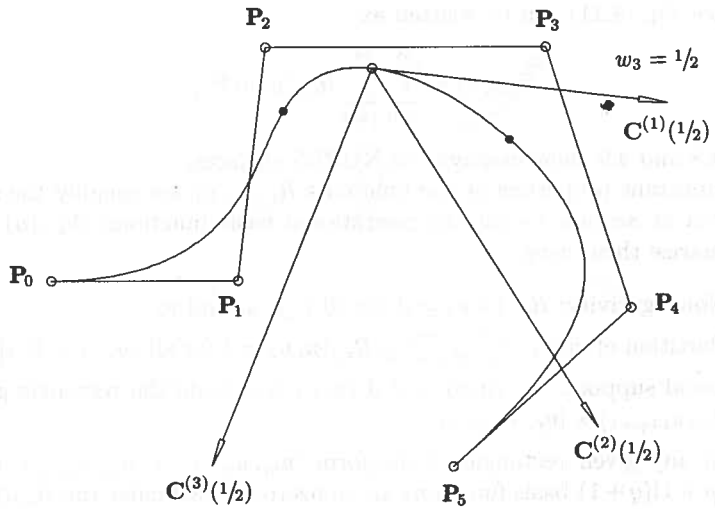


Figure 4.7. First, second, and third derivatives of a cubic NURBS curve computed at $u = 1/2$, with $w_3 = 1/2$ and $w_i = 1, i \neq 3$.

4.4 Definition and Properties of NURBS Surfaces

A NURBS surface of degree p in the u direction and degree q in the v direction is a bivariate vector-valued piecewise rational function of the form

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \qquad 0 \le u,v \le 1 \tag{4.11}$$

The $\{\mathbf{P}_{i,j}\}$ form a bidirectional control net, the $\{w_{i,j}\}$ are the weights, and the $\{N_{i,p}(u)\}$ and $\{N_{j,q}(v)\}$ are the nonrational B-spline basis functions defined on the knot vectors

$$U = \{\underbrace{0,\ldots,0}_{p+1}, u_{p+1},\ldots,u_{r-p-1}, \underbrace{1,\ldots,1}_{p+1}\}$$

$$V = \{\underbrace{0,\ldots,0}_{q+1}, v_{q+1},\ldots,v_{s-q-1}, \underbrace{1,\ldots,1}_{q+1}\}$$

where $r = n+p+1$ and $s = m+q+1$.

Introducing the piecewise rational basis functions

$$R_{i,j}(u,v) = \frac{N_{i,p}(u)N_{j,q}(v)w_{i,j}}{\sum_{k=0}^{n}\sum_{l=0}^{m} N_{k,p}(u)N_{l,q}(v)w_{k,l}} \tag{4.12}$$

the surface Eq. (4.11) can be written as

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}(u,v)\mathbf{P}_{i,j} \tag{4.13}$$

Figures 4.8 and 4.9 show examples of NURBS surfaces.

The important properties of the functions $R_{i,j}(u,v)$ are roughly the same as those given in Section 3.4 for the nonrational basis functions, $N_{i,p}(u)\,N_{j,q}(v)$. We summarize them here.

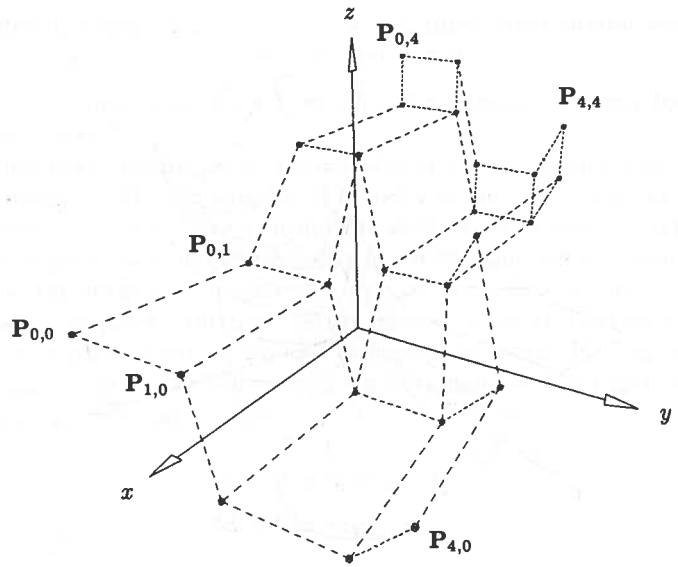
P4.15 Nonnegativity: $R_{i,j}(u,v) \ge 0$ for all i, j, u, and v;

P4.16 Partition of unity: $\sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}(u,v) = 1$ for all $(u,v) \in [0,1]\times[0,1]$;

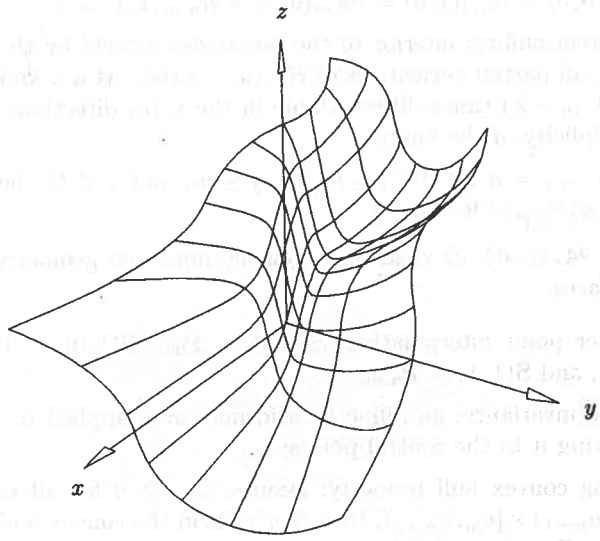
P4.17 Local support: $R_{i,j}(u,v) = 0$ if (u,v) is outside the rectangle given by $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$;

P4.18 In any given rectangle of the form $[u_{i_0}, u_{i_0+1}) \times [v_{j_0}, v_{j_0+1})$, at most $(p+1)(q+1)$ basis functions are nonzero, in particular the $R_{i,j}(u,v)$ for $i_0 - p \le i \le i_0$ and $j_0 - q \le j \le j_0$ are nonzero;

P4.19 Extrema: if $p > 0$ and $q > 0$, then $R_{i,j}(u,v)$ attains exactly one maximum value;



(a)



(b)

Figure 4.8. Control net and biquadratic NURBS surface, $w_{1,1} = w_{1,2} = w_{2,1} = w_{2,2} = 10$, with the rest of the weights 1. $U = V = \{0,0,0,\frac{1}{3},\frac{2}{3},1,1,1\}$. (a) Control net; (b) biquadratic NURBS surface.

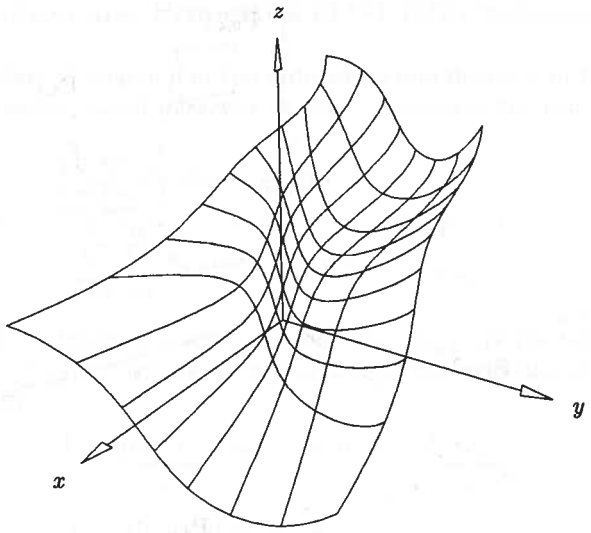


Figure 4.9. Bicubic NURBS surface defined by the control net in Figure 4.8a, with $U = V = \{0, 0, 0, 0, \frac{1}{2}, 1, 1, 1, 1\}$ and with the same weights as in Figures 4.8.

P4.20 $R_{0,0}(0,0) = R_{n,0}(1,0) = R_{0,m}(0,1) = R_{n,m}(1,1) = 1$;

P4.21 Differentiability: interior to the rectangles formed by the u and v knot lines, all partial derivatives of $R_{i,j}(u,v)$ exist. At a u knot (v knot) it is $p-k$ ($q-k$) times differentiable in the u (v) direction, where k is the multiplicity of the knot;

P4.22 If all $w_{i,j} = a$ for $0 \le i \le n$, $0 \le j \le m$, and $a \neq 0$, then $R_{i,j}(u,v) = N_{i,p}(u)N_{j,q}(v)$ for all i, j.

Properties P4.15–P4.22 yield the following important geometric properties of NURBS surfaces:

P4.23 Corner point interpolation: $\mathbf{S}(0,0) = \mathbf{P}_{0,0}$, $\mathbf{S}(1,0) = \mathbf{P}_{n,0}$, $\mathbf{S}(0,1) = \mathbf{P}_{0,m}$, and $\mathbf{S}(1,1) = \mathbf{P}_{n,m}$;

P4.24 Affine invariance: an affine transformation is applied to the surface by applying it to the control points;

P4.25 Strong convex hull property: assume $w_{i,j} \ge 0$ for all i, j. If $(u,v) \in [u_{i_0}, u_{i_0+1}) \times [v_{j_0}, v_{j_0+1})$, then $\mathbf{S}(u,v)$ is in the convex hull of the control points $\mathbf{P}_{i,j}$, $i_0 - p \le i \le i_0$ and $j_0 - q \le j \le j_0$;

P4.26 Local modification: if $\mathbf{P}_{i,j}$ is moved, or $w_{i,j}$ is changed, it affects the surface shape only in the rectangle $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$;

P4.27 Nonrational B-spline and Bézier and rational Bézier surfaces are special cases of NURBS surfaces;

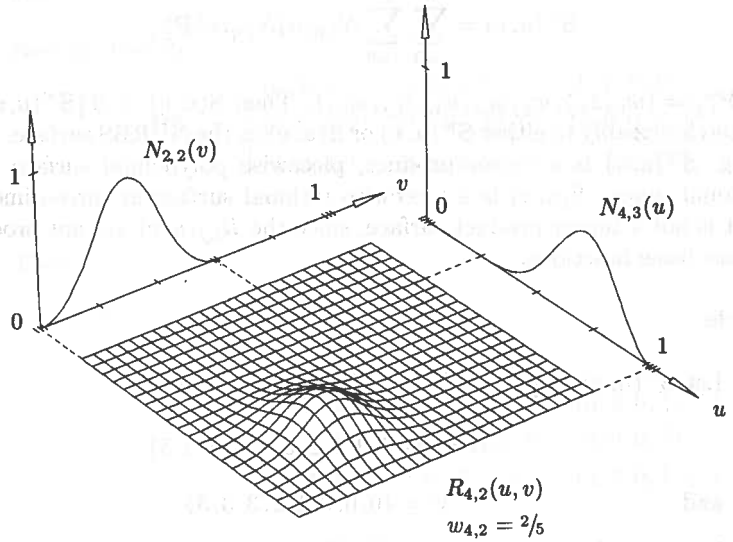
P4.28 Differentiability: $\mathbf{S}(u,v)$ is $p-k$ ($q-k$) times differentiable with respect to u (v) at a u knot (v knot) of multiplicity k.

We remark that there is no known variation diminishing property for NURBS surfaces (see [Prau92]).

We can use both control point movement and weight modification to locally change the shape of NURBS surfaces. Figures 4.10 and 4.11 show the effects on the basis function $R_{i,j}(u,v)$ and the surface shape when a single weight, $w_{i,j}$, is modified. Compare these figures with Figures 3.19b and 3.20a,b. Qualitatively, the effect on the surface is: Assume $(u,v) \in [u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$; then if $w_{i,j}$ increases (decreases), the point $\mathbf{S}(u,v)$ moves closer to (farther from) $\mathbf{P}_{i,j}$, and hence the surface is pulled toward (pushed away from) $\mathbf{P}_{i,j}$. As is the case for curves, the movement of $\mathbf{S}(u,v)$ is along a straight line. In Figure 4.12 (u,v) are fixed and $w_{2,2}$ is changing. Let

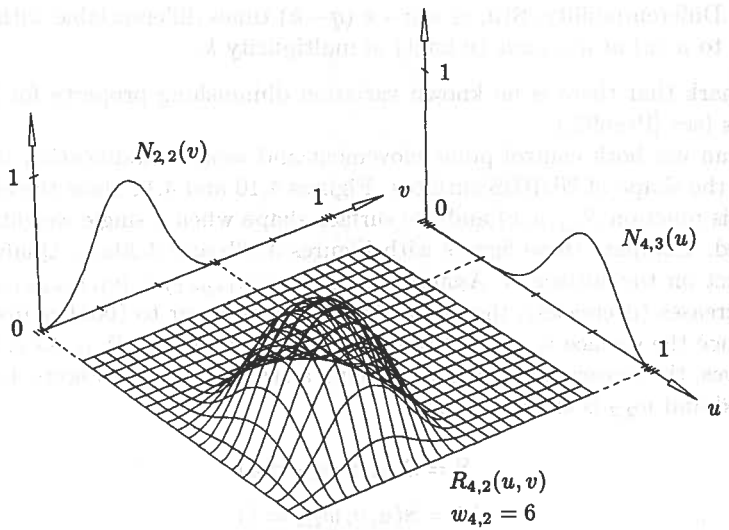
$$\mathbf{S} = \mathbf{S}(u, v; w_{2,2} = 0)$$
$$\mathbf{M} = \mathbf{S}(u, v; w_{2,2} = 1) \tag{4.14}$$

Then the straight line defined by $\mathbf{S}$ and $\mathbf{M}$ passes through $\mathbf{P}_{2,2}$, and for arbitrary $w_{2,2}$, $0 < w_{2,2} < \infty$, $\mathbf{S}_{2,2} = \mathbf{S}(u, v; w_{2,2})$ lies on this line segment between $\mathbf{S}$ and $\mathbf{P}_{2,2}$.



(a)

Figure 4.10. The basis function $R_{4,2,}(u,v)$, with $U = \{0, 0, 0, 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, 1, 1, 1\}$ and $V = \{0, 0, 0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{3}{5}, \frac{4}{5}, 1, 1, 1\}$. $w_{i,j} = 1$ for all $(i,j) \neq (4,2)$. (a) $w_{4,2} = \frac{2}{5}$; (b) $w_{4,2} = 6$.



(b)

Figure 4.10. (*Continued.*)

It is convenient to represent a NURBS surface using homogeneous coordinates, that is

$$\mathbf{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\mathbf{P}^w_{i,j} \tag{4.15}$$

where $\mathbf{P}^w_{i,j} = (w_{i,j}x_{i,j}, w_{i,j}y_{i,j}, w_{i,j}z_{i,j}, w_{i,j})$. Then $\mathbf{S}(u,v) = H\{\mathbf{S}^w(u,v)\}$. We refer interchangeably to either $\mathbf{S}^w(u,v)$ or $\mathbf{S}(u,v)$ as the NURBS surface. Strictly speaking, $\mathbf{S}^w(u,v)$ is a tensor product, piecewise polynomial surface in four-dimensional space. $\mathbf{S}(u,v)$ is a piecewise rational surface in three-dimensional space; it is not a tensor product surface, since the $R_{i,j}(u,v)$ are not products of univariate basis functions.

Example

Ex4.3 Let $\mathbf{S}^w(u,v) = \sum_{i=0}^{7}\sum_{j=0}^{4} N_{i,2}(u)N_{j,2}(v)$, with

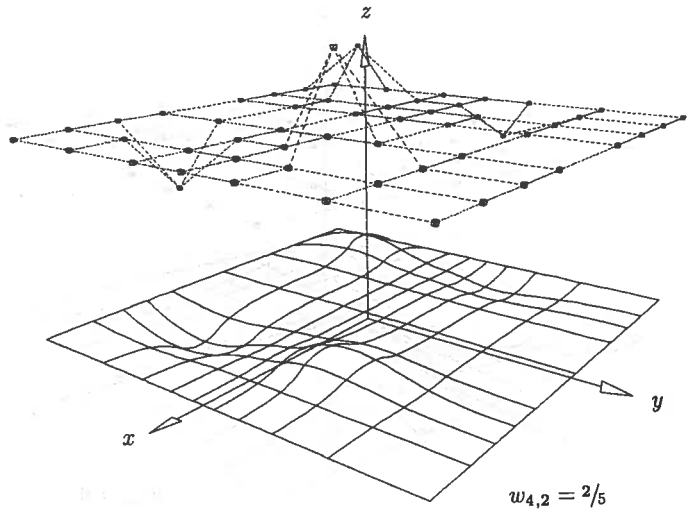
$$U = \{0,0,0,1,2,3,4,4,5,5,5\}$$

and

$$V = \{0,0,0,1,2,3,3,3\}$$

Let us evaluate the surface at $(u,v) = (5/2, 1)$. Then $u \in [u_4, u_5)$ and $v \in [v_3, v_4)$, and from Sections 3.2 and 4.2 we know that

$$N_{2,2}\left(\frac{5}{2}\right) = \frac{1}{8} \quad N_{3,2}\left(\frac{5}{2}\right) = \frac{6}{8} \quad N_{4,2}\left(\frac{5}{2}\right) = \frac{1}{8}$$



(a)

Figure 4.11. Cubic × quadratic surfaces corresponding to Figure 4.10, with the control net offset for better visualization. (a) $w_{4,2} = 2/5$; (b) $w_{4,2} = 6$.

and

$$N_{1,2}(1) = \frac{1}{2} \quad N_{2,2}(1) = \frac{1}{2} \quad N_{3,2}(1) = 0$$

Now assume that

$$[\,\mathbf{P}^w_{i,j}\,] = \begin{bmatrix} (0,2,4,1) & (0,6,4,2) & (0,2,0,1) \\ (4,6,8,2) & (12,24,12,6) & (4,6,0,2) \\ (4,2,4,1) & (8,6,4,2) & (4,2,0,1) \end{bmatrix}$$

$$i = 2,3,4; \quad j = 1,2,3$$

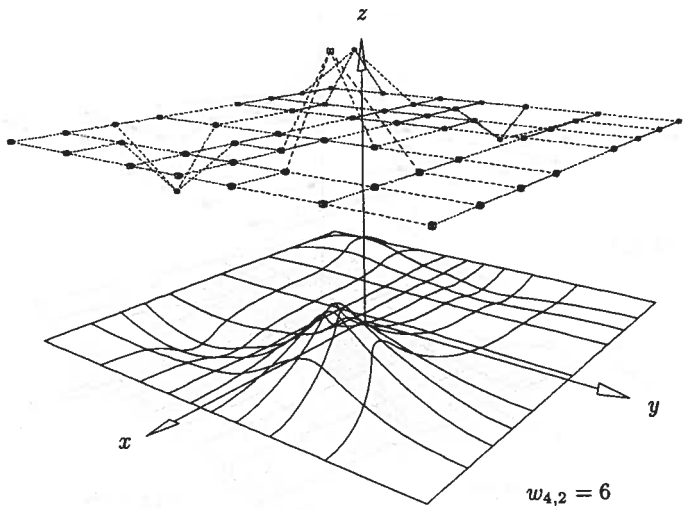
Then

$$\mathbf{S}^w\left(\frac{5}{2},1\right) = \begin{bmatrix} \frac{1}{8} & \frac{6}{8} & \frac{1}{8} \end{bmatrix} \times \begin{bmatrix} (0,2,4,1) & (0,6,4,2) & (0,2,0,1) \\ (4,6,8,2) & (12,24,12,6) & (4,6,0,2) \\ (4,2,4,1) & (8,6,4,2) & (4,2,0,1) \end{bmatrix} \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix}$$

$$= \left(\frac{54}{8}, \frac{98}{8}, \frac{68}{8}, \frac{27}{8}\right)$$

Projecting yields

$$\mathbf{S}\left(\frac{5}{2},1\right) = \left(2, \frac{98}{27}, \frac{68}{27}\right)$$



(b)

Figure 4.11. (*Continued.*)

Algorithm A3.5 can be adapted to compute a point on a rational B-spline surface by simply allowing the array P to contain weighted control points (use Pw), accumulating the four-dimensional surface point in Sw and inserting the line S = Sw/w at the end of the algorithm.

```
ALGORITHM A4.3
  SurfacePoint(n,p,U,m,q,V,Pw,u,v,S)
    { /*  Compute point on rational B-spline surface  */
      /*  Input:  n,p,U,m,q,V,Pw,u,v  */
      /*  Output: S  */
    uspan = FindSpan(n,p,u,U);
    BasisFuns(uspan,u,p,U,Nu);
    vspan = FindSpan(m,q,v,V);
    BasisFuns(vspan,v,q,V,Nv);
    for (l=0; l<=q; l++)
      {
      temp[l] = 0.0;
      for (k=0; k<=p; k++)
        temp[l] = temp[l] + Nu[k]*Pw[uspan-p+k][vspan-q+l];
      }
    Sw = 0.0;
    for (l=0; l<=q; l++)
      Sw = Sw + Nv[l]*temp[l];
    S = Sw/w;
    }
```

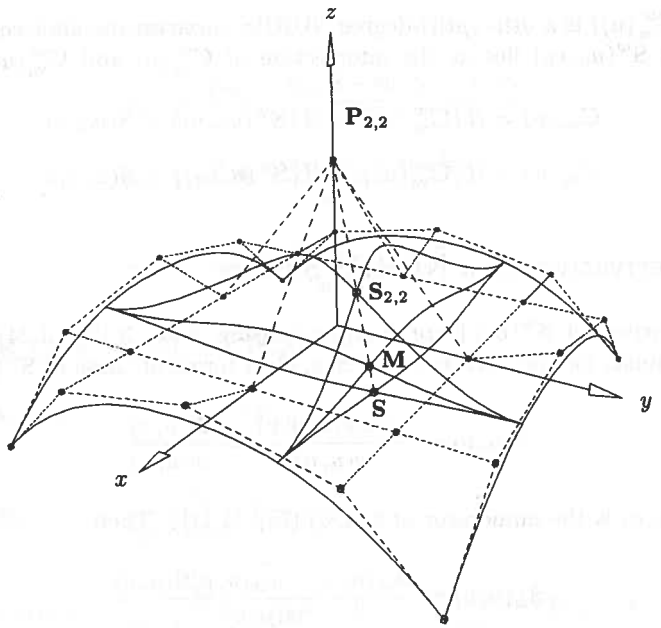



Figure 4.12. Modification of the weight $w_{2,2}$.

By applying Eqs. (3.14) and (3.15), we obtain isoparametric curves on a NURBS surface. First, fix $u = u_0$

$$\mathbf{C}^w_{u_0}(v) = \mathbf{S}^w(u_0, v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u_0) N_{j,q}(v)\, \mathbf{P}^w_{i,j} \tag{4.16}$$

$$= \sum_{j=0}^{m} N_{j,q}(v) \left(\sum_{i=0}^{n} N_{i,p}(u_0)\, \mathbf{P}^w_{i,j} \right)$$

$$= \sum_{j=0}^{m} N_{j,q}(v)\, \mathbf{Q}^w_j(u_0)$$

where $$\mathbf{Q}^w_j(u_0) = \sum_{i=0}^{n} N_{i,p}(u_0)\, \mathbf{P}^w_{i,j}$$

Analogously $$\mathbf{C}^w_{v_0}(u) = \sum_{i=0}^{n} N_{i,p}(u)\, \mathbf{Q}^w_i(v_0) \tag{4.17}$$

where $$\mathbf{Q}^w_i(v_0) = \sum_{j=0}^{m} N_{j,q}(v_0)\, \mathbf{P}^w_{i,j}$$

$\mathbf{C}^w_{u_0}(v)$ ($\mathbf{C}^w_{v_0}(u)$) is a qth- (pth)-degree NURBS curve on the knot vector V (U). The point $\mathbf{S}^w(u_0, v_0)$ lies at the intersection of $\mathbf{C}^w_{u_0}(v)$ and $\mathbf{C}^w_{v_0}(u)$. Projecting yields

$$\mathbf{C}_{u_0}(v) = H\{\mathbf{C}^w_{u_0}(v)\} = H\{\mathbf{S}^w(u_0, v)\} = \mathbf{S}(u_0, v)$$

$$\mathbf{C}_{v_0}(u) = H\{\mathbf{C}^w_{v_0}(u)\} = H\{\mathbf{S}^w(u, v_0)\} = \mathbf{S}(u, v_0) \tag{4.18}$$

4.5 Derivatives of a NURBS Surface

The derivatives of $\mathbf{S}^w(u, v)$ are computed using Eqs. (3.17)–(3.24). We now derive formulas for the derivatives of $\mathbf{S}(u, v)$ in terms of those of $\mathbf{S}^w(u, v)$. Let

$$\mathbf{S}(u, v) = \frac{w(u, v)\mathbf{S}(u, v)}{w(u, v)} = \frac{\mathbf{A}(u, v)}{w(u, v)}$$

where $\mathbf{A}(u, v)$ is the numerator of $\mathbf{S}(u, v)$ (Eq. [4.11]). Then

$$\mathbf{S}_\alpha(u, v) = \frac{\mathbf{A}_\alpha(u, v) - w_\alpha(u, v)\mathbf{S}(u, v)}{w(u, v)} \tag{4.19}$$

where α denotes either u or v.

In general

$$\mathbf{A}^{(k,l)} = \left[(w\mathbf{S})^k\right]^l = \left(\sum_{i=0}^{k}\binom{k}{i} w^{(i,0)}\mathbf{S}^{(k-i,0)}\right)^l$$

$$= \sum_{i=0}^{k}\binom{k}{i}\sum_{j=0}^{l}\binom{l}{j} w^{(i,j)}\mathbf{S}^{(k-i,l-j)}$$

$$= w^{(0,0)}\mathbf{S}^{(k,l)} + \sum_{i=1}^{k}\binom{k}{i} w^{(i,0)}\mathbf{S}^{(k-i,l)} + \sum_{j=1}^{l}\binom{l}{j} w^{(0,j)}\mathbf{S}^{(k,l-j)}$$

$$+ \sum_{i=1}^{k}\binom{k}{i}\sum_{j=1}^{l}\binom{l}{j} w^{(i,j)}\mathbf{S}^{(k-i,l-j)}$$

and it follows that

$$\mathbf{S}^{(k,l)} = \frac{1}{w}\left(\mathbf{A}^{(k,l)} - \sum_{i=1}^{k}\binom{k}{i} w^{(i,0)}\mathbf{S}^{(k-i,l)}\right.$$

$$\left. - \sum_{j=1}^{l}\binom{l}{j} w^{(0,j)}\mathbf{S}^{(k,l-j)} - \sum_{i=1}^{k}\binom{k}{i}\sum_{j=1}^{l}\binom{l}{j} w^{(i,j)}\mathbf{S}^{(k-i,l-j)}\right) \tag{4.20}$$

From Eq. (4.20) we obtain

$$\mathbf{S}_{uv} = \frac{\mathbf{A}_{uv} - w_{uv}\mathbf{S} - w_u\mathbf{S}_v - w_v\mathbf{S}_u}{w} \tag{4.21}$$

$$\mathbf{S}_{uu} = \frac{\mathbf{A}_{uu} - 2w_u\mathbf{S}_u - w_{uu}\mathbf{S}}{w} \tag{4.22}$$

$$\mathbf{S}_{vv} = \frac{\mathbf{A}_{vv} - 2w_v\mathbf{S}_v - w_{vv}\mathbf{S}}{w} \tag{4.23}$$

From Eqs. (3.24), (4.19), and (4.20)

$$\mathbf{S}_u(0,0) = \frac{p}{u_{p+1}}\frac{w_{1,0}}{w_{0,0}}(\mathbf{P}_{1,0} - \mathbf{P}_{0,0}) \tag{4.24}$$

$$\mathbf{S}_v(0,0) = \frac{q}{v_{q+1}}\frac{w_{0,1}}{w_{0,0}}(\mathbf{P}_{0,1} - \mathbf{P}_{0,0}) \tag{4.25}$$

$$\mathbf{S}_{uv}(0,0) = \frac{pq}{w_{0,0}u_{p+1}v_{q+1}}\left(w_{1,1}\mathbf{P}_{1,1} - \frac{w_{1,0}w_{0,1}}{w_{0,0}}(\mathbf{P}_{1,0} + \mathbf{P}_{0,1})\right.$$

$$\left. + \left(\frac{2w_{1,0}w_{0,1}}{w_{0,0}} - w_{1,1}\right)\mathbf{P}_{0,0}\right) \tag{4.26}$$

Figure 4.13 shows the first- and second-order partial derivatives of a NURBS surface. The first partials are scaled down by $1/2$, and the second partials are scaled down by $1/3$.

Now assume that (u, v) is fixed, and that all derivatives $\mathbf{A}^{(k,l)}$, $w^{(k,l)}$ for $k, l \geq 0$ and $0 \leq k + l \leq d$, have been computed and loaded into the arrays `Aders` and `wders`, respectively. Algorithm `A4.4` computes the point $\mathbf{S}(u, v)$ and the derivatives $\mathbf{S}^{(k,l)}(u, v)$, $0 \leq k + l \leq d$. `Bin[][]` contains the precomputed binomial coefficients.

```
ALGORITHM A4.4
  RatSurfaceDerivs(Aders,wders,d,SKL)
    { /*  Compute S(u,v) derivatives  */
      /*  from Sw(u,v) derivatives  */
      /*  Input:  Aders,wders,d  */
      /*  Output: SKL  */
    for (k=0; k<=d; k++)
      for (l=0; l<=d-k; l++)
        {
        v = Aders[k][l];
        for (j=1; j<=l; j++)
          v = v - Bin[l][j]*wders[0][j]*SKL[k][l-j];
```

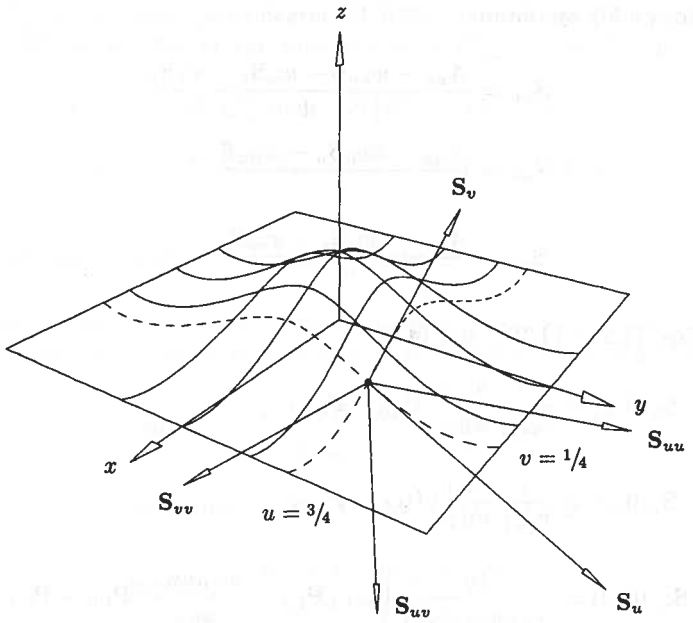


Figure 4.13. The first- and second-order partial derivatives of a bicubic NURBS surface computed at $u = 3/4$ and $v = 1/4$.

```
        for (i=1; i<=k; i++)
          {
          v = v - Bin[k][i]*wders[i][0]*SKL[k-i][l];
          v2 = 0.0;
          for (j=1; j<=l; j++)
            v2 = v2 + Bin[l][j]*wders[i][j]*SKL[k-i][l-j];
          v = v - Bin[k][i]*v2;
          }
        SKL[k][l] = v/wders[0][0];
        }
    }
```

Exercises

4.1. Let $U = \{0,0,0,1/3,2/3,1,1,1\}$ and $\{w_0,\ldots,w_4\}=\{1,4,1,1,1\}$. Using the Cox-deBoor recurrence formula (Eq. [2.5]) and Eq. (4.2), compute the five quadratic rational functions, $R_{i,2}(u)$, $0 \le i \le 4$. The graphs of these functions are shown in Figure 4.5a. Assume $\{\mathbf{P}_0,\ldots,\mathbf{P}_4\} = \{(0,0),(1,1),(3,2),(4,1),(5,-1)\}$ are control points in the xy plane. Compute the rational coordinate functions $x(u)$ and $y(u)$ representing $\mathbf{C}(u)$ in the interval $u \in [1/3, 2/3)$.

4.2. Refer to Example Ex4.2 for a quadratic rational Bézier circular arc; compute $\mathbf{C}''(1)$.

4.3. Let $\mathbf{C}^w(u) = \sum_{i=0}^{1} N_{i,1}(u)\mathbf{P}_i^w$ be a line segment in the xy plane, where $\mathbf{P}_0 = (0,1)$, $\mathbf{P}_1 = (2,0)$, $w_0 = 1$, $w_1 = 3$, and $U = \{0,0,1,1\}$. Derive the rational functions representing the x and y coordinates of this line segment, i.e., $x(u)$ and $y(u)$, where $\mathbf{C}(u) = (x(u), y(u))$. Compute $\mathbf{C}'(0)$ using Eq. (4.9) and $\mathbf{C}''(0)$ using Eqs. (4.8) and (3.9). Then set $w_1 = 1$ and recompute $x(u)$, $y(u)$, $\mathbf{C}'(0)$, and $\mathbf{C}''(0)$.

4.4. Let $\mathbf{S}^w(u,v) = \sum_{i=0}^{1}\sum_{j=0}^{1} N_{i,1}(u)N_{j,1}(v)\mathbf{P}_{i,j}^w$, where $\{\mathbf{P}_{0,0}, \mathbf{P}_{1,0}, \mathbf{P}_{0,1}, \mathbf{P}_{1,1}\} = \{(0,0,1),(0,1,3),(2,1,1),(2,0,3)\}$, $\{w_{0,0}, w_{1,0}, w_{0,1}, w_{1,1}\} = \{2,1,1,1\}$, and $U = V = \{0,0,1,1\}$. Derive the four rational basis functions, $R_{i,j}(u,v)$, $0 \le i,j \le 1$, and the rational coordinate functions $x(u,v)$, $y(u,v)$, and $z(u,v)$ of the surface $\mathbf{S}(u,v)$.

4.5. From $\mathbf{S}^w(u,v)$ in Exercise 4.4 derive the two rational isoparametric curves $\mathbf{C}_{u_0}^w(v)$ and $\mathbf{C}_{v_0}^w(u)$, for $u_0 = 1/3$ and $v_0 = 1/2$. Then evaluate the curves $\mathbf{C}_{u_0}(v)$ and $\mathbf{C}_{v_0}(u)$ at $v = 1/2$ and $u = 1/3$, respectively. Check your results by substituting $(u,v) = (1/3, 1/2)$ into the rational coordinate functions obtained in Exercise 4.4.

4.6. Let $\mathbf{S}^w(u,v) = \sum_{i=0}^{1}\sum_{j=0}^{1} N_{i,1}(u)N_{j,1}(v)\mathbf{P}_{i,j}^w$ be the surface given in Exercise 4.4. Since $N_{0,1}(1/2) = N_{1,1}(1/2) = 1/2$, it follows that

$$\mathbf{S}^w\left(\frac{1}{2},\frac{1}{2}\right) = \frac{1}{4}\left(\mathbf{P}_{0,0}^w + \mathbf{P}_{0,1}^w + \mathbf{P}_{1,0}^w + \mathbf{P}_{1,1}^w\right) = \left(1, \frac{1}{2}, \frac{9}{4}, \frac{5}{4}\right)$$

Compute $\mathbf{S}_u(1/2, 1/2)$, $\mathbf{S}_v(1/2, 1/2)$, $\mathbf{S}_{uv}(1/2, 1/2)$, $\mathbf{S}_{uu}(1/2, 1/2)$.

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

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

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

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

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.

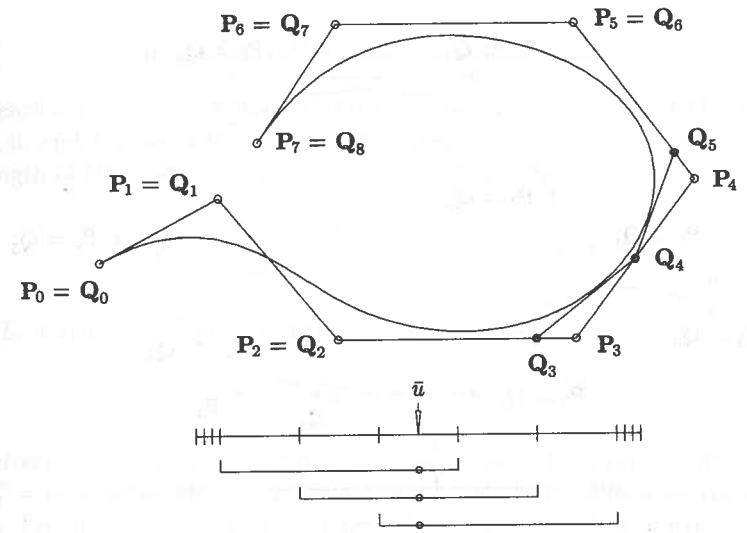
Ex5.2 Use the same curve as in Ex5.1, that is, $p = 3$, $U = \{0,0,0,0,1,2,3,4,5,5,5,5\}$, and $\mathbf{P}_0,\ldots,\mathbf{P}_7$. This time we insert a knot which is already in the knot vector, namely $\bar{u} = 2$. Then $\bar{u} \in [u_5, u_6)$, $k = 5$, and again we compute $\mathbf{Q}_3$, $\mathbf{Q}_4$,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

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

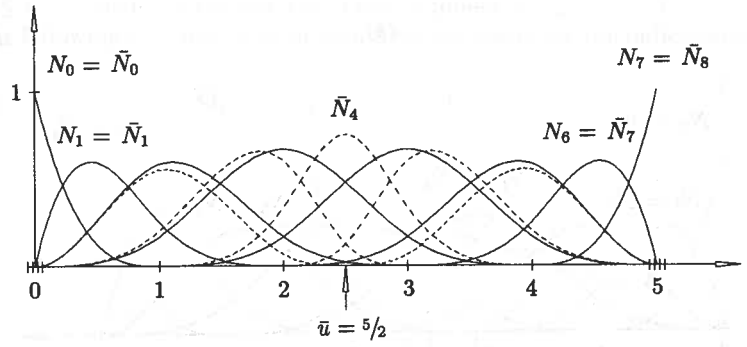
$$\alpha_4 = \frac{2-1}{4-1} = \frac{1}{3} \implies \mathbf{Q}_4 = \frac{1}{3}\mathbf{P}_4 + \frac{2}{3}\mathbf{P}_3$$

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

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)

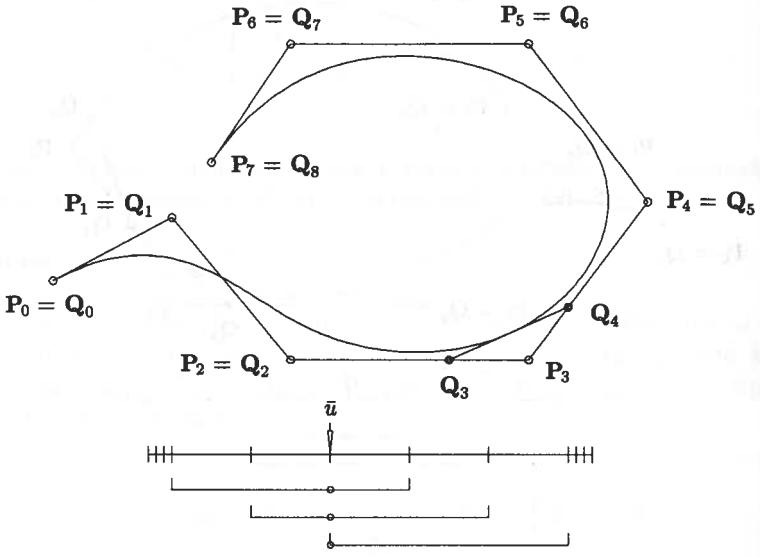


(b)

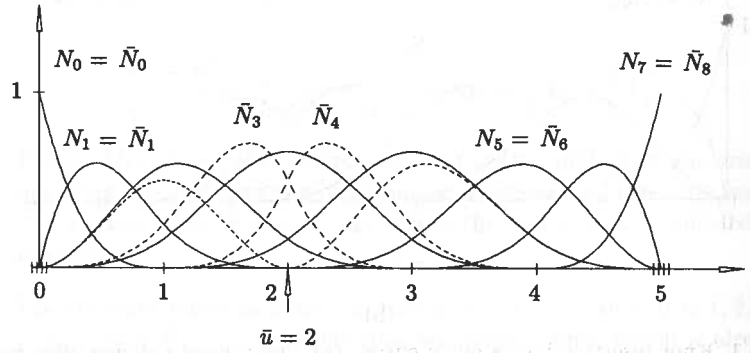
Figure 5.1. Knot insertion into a cubic curve. (a) The control polygon after inserting $u = 5/2$ into the knot vector $\{0,0,0,0,1,2,3,4,5,5,5,5\}$; (b) the original (solid) and the new (dashed) basis functions before and after knot insertion.

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

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

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 $[u_i, u_{i+p})$

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

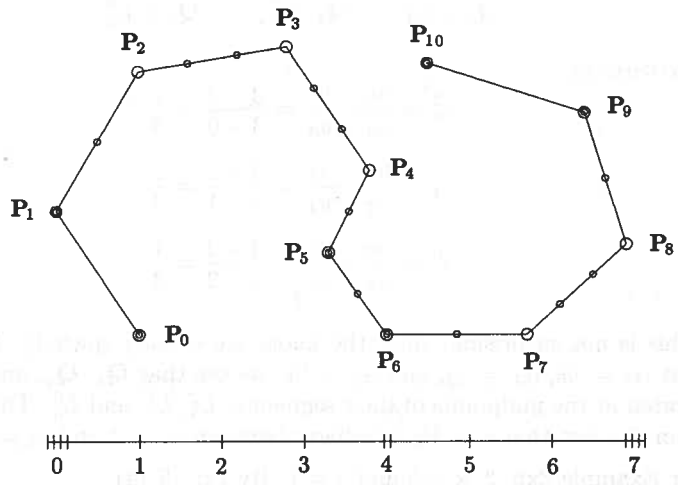
Then L_i is partitioned into segments L_i^j, whose lengths are

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

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.

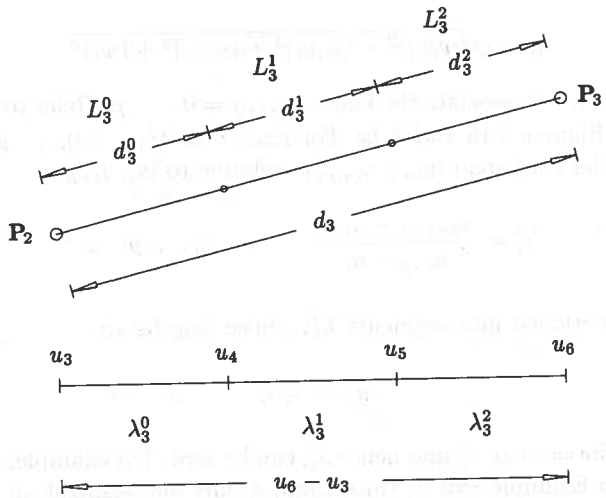
Now Figures 5.3a and 5.3b and Eq. (5.13) show how the polygon legs are subdivided, and Figures 5.1a and 5.2a show how polygon corners are cut in the knot insertion process. In particular, if $\bar{u} \in [u_k, u_{k+1})$ is a knot of multiplicity s, $0 \le s \le p-1$, then the corners with control points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

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

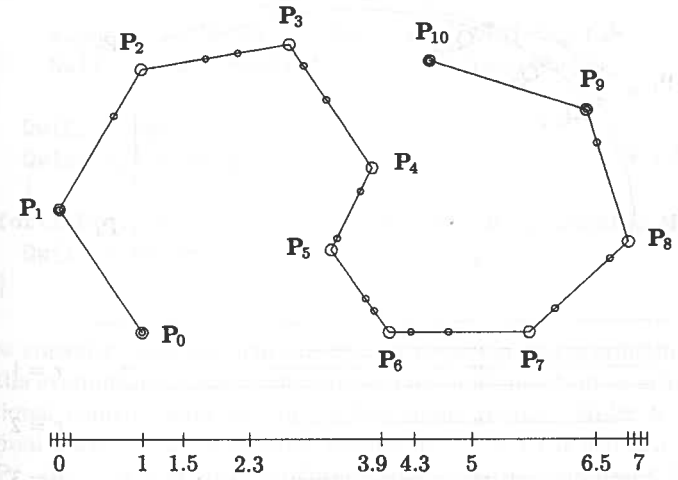


Figure 5.4. Nonuniform partitioning of the control polygon in Figure 5.3a defined by the knot vector $\{0,0,0,0,1,1.5,2.3,3.9,4.3,5,6.5,7,7,7,7\}$.

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

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

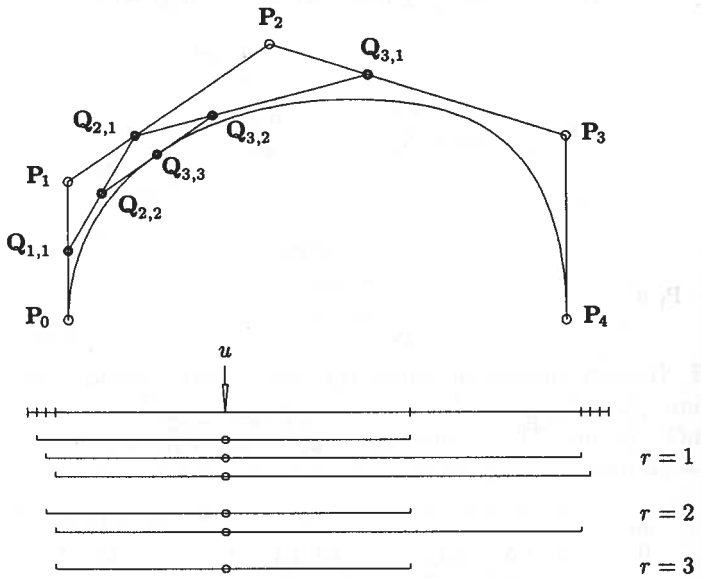


Figure 5.5. Knot insertion into a cubic curve three times. The curve is defined on $\{0,0,0,0,2,3,3,3,3\}$.

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; 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] = Rw[p-j];
    }
  for(i=L+1; i<k; 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.

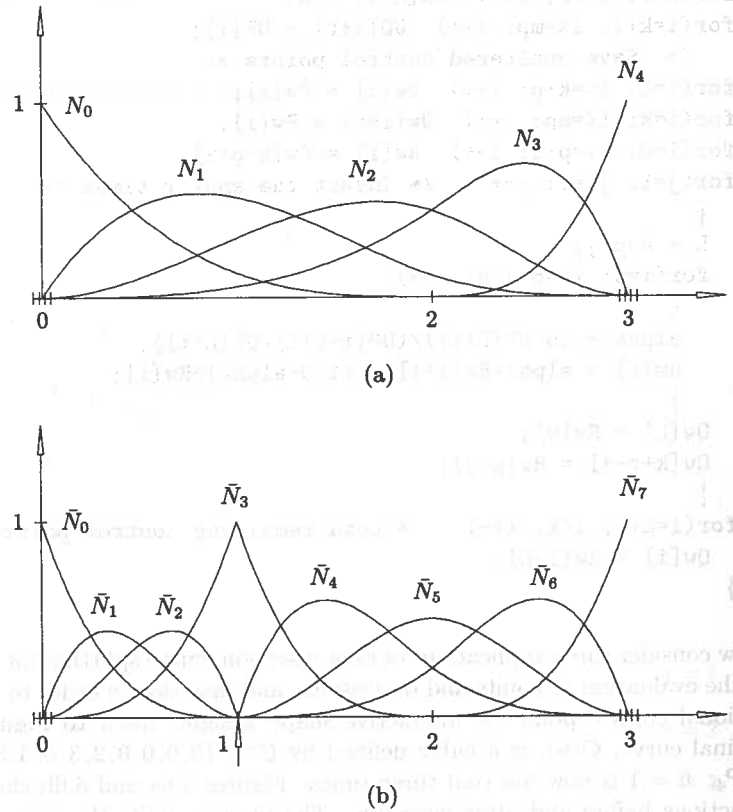


Figure 5.6. Basis functions corresponding to Figure 5.5. (a) Before knot insertion; (b) after knot insertion.

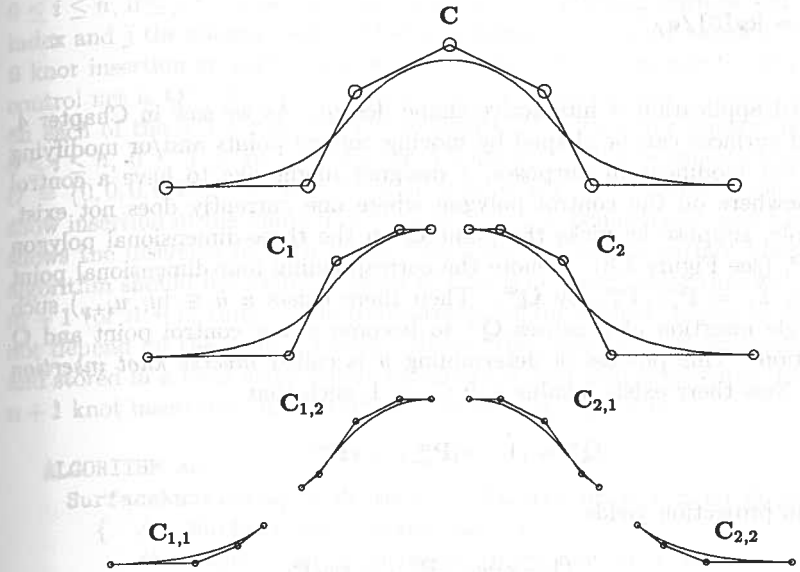


Figure 5.7. Splitting a cubic curve defined on $\{0,0,0,0,1/4,1/2,3/4,1,1,1,1\}$.

```
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 $\mathbf{P}^w_{i,j}$,

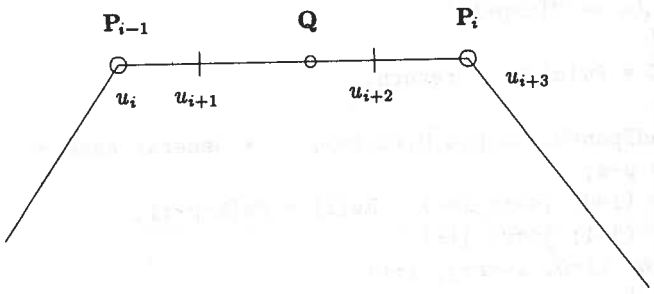


Figure 5.8. Inverse knot insertion for cubic curves; the point $\mathbf{Q}$ lies on the leg $\mathbf{P}_{i-1}\mathbf{P}_i$ partitioned by the knots u_i, u_{i+1}, u_{i+2}, and u_{i+3}.

$0 \le i \le n$, $0 \le j \le m$, 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)
```

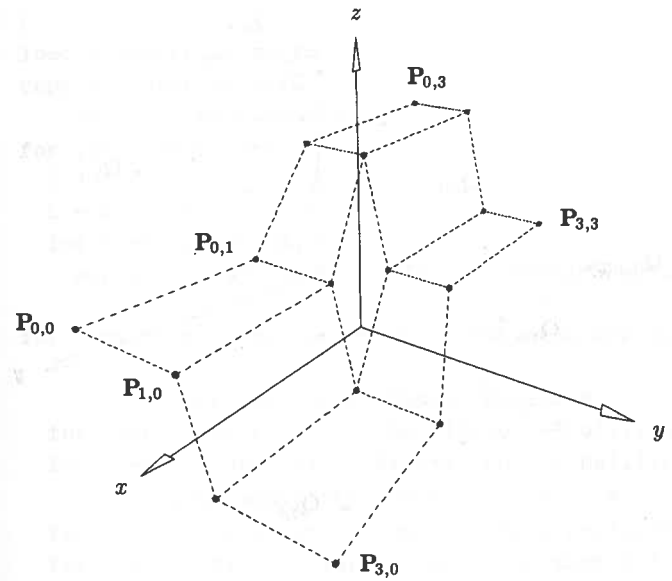
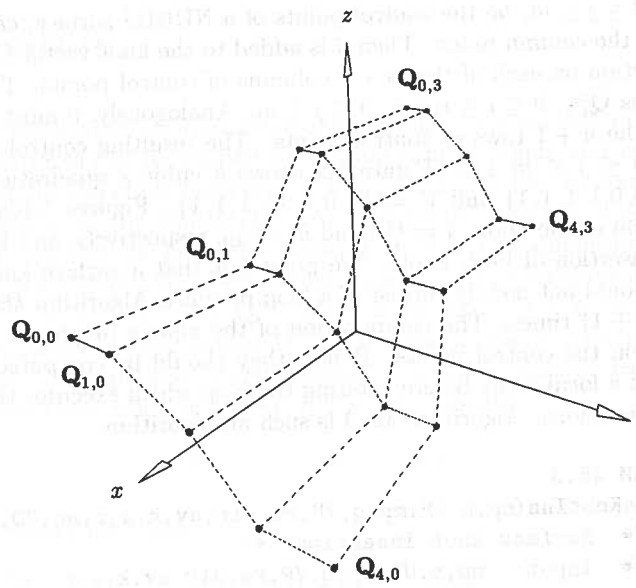
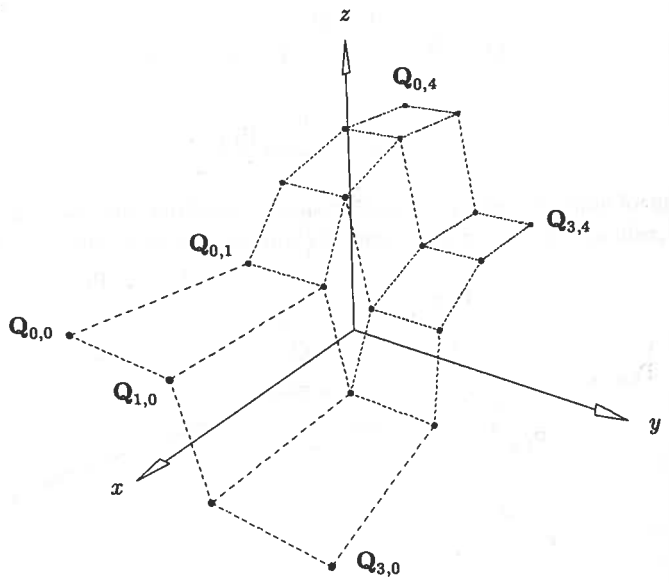


Figure 5.9. The control net of a (cubic × quadratic) surface defined on the knot vectors $U = \{0,0,0,0,1,1,1,1\}$ and $V = \{0,0,0,1/2,1,1,1\}$.

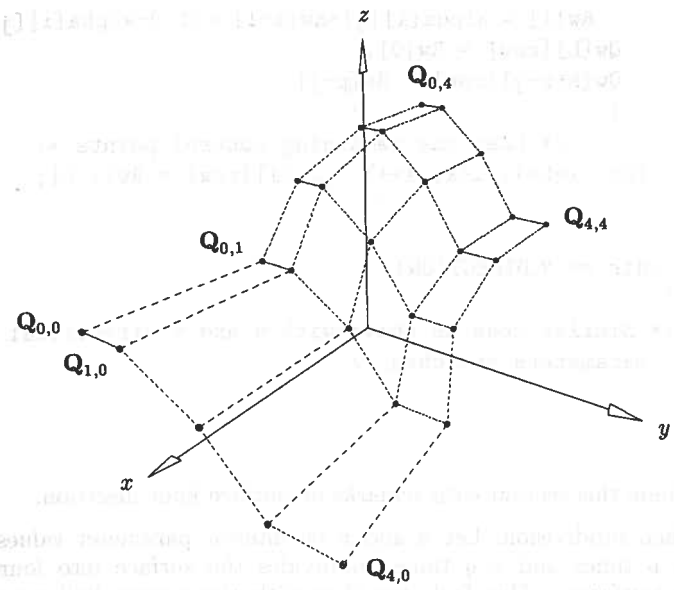


(a)



(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; 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][row] = Rw[p-j];
          }
              /* Load the remaining control points */
        for (i=L+1; i<k; 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;

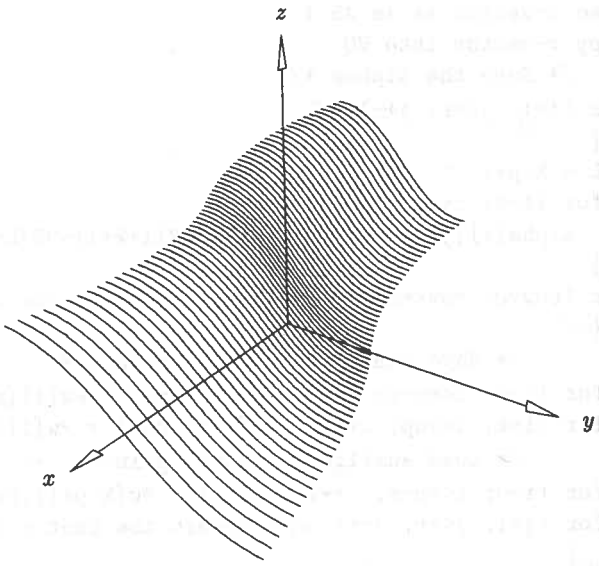
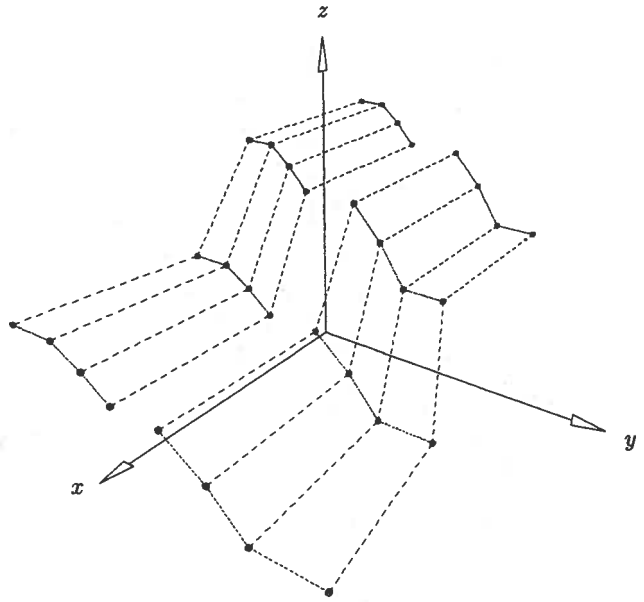
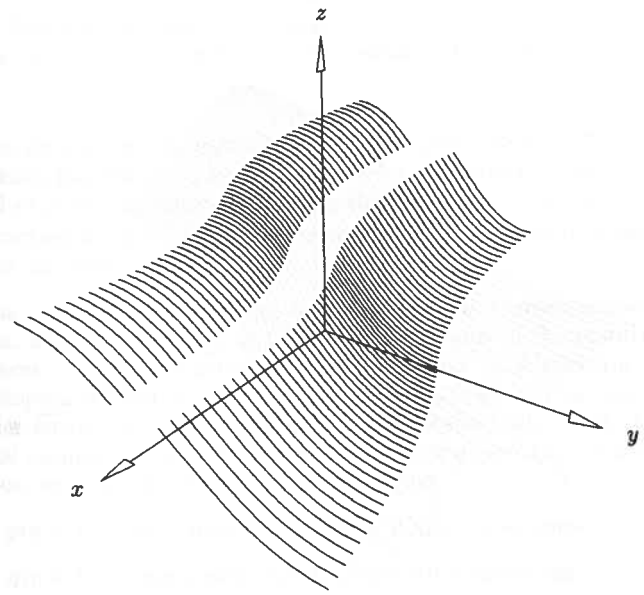


Figure 5.11. A (cubic × quadratic) surface defined as in Figure 5.9.

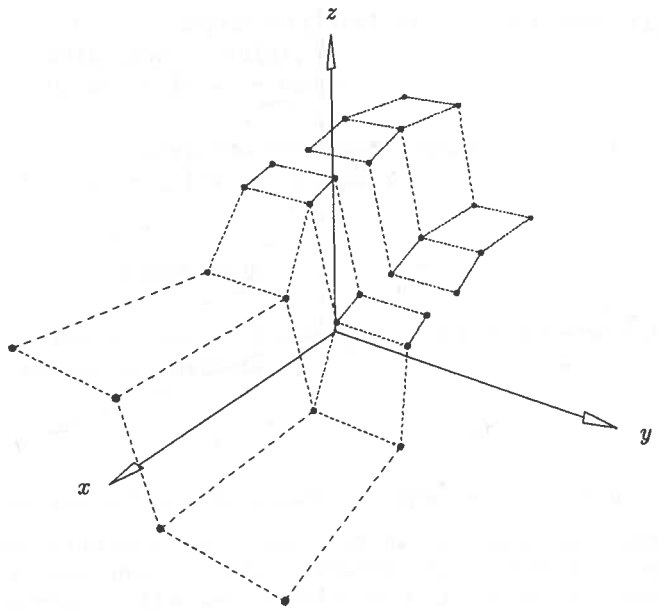


(a)

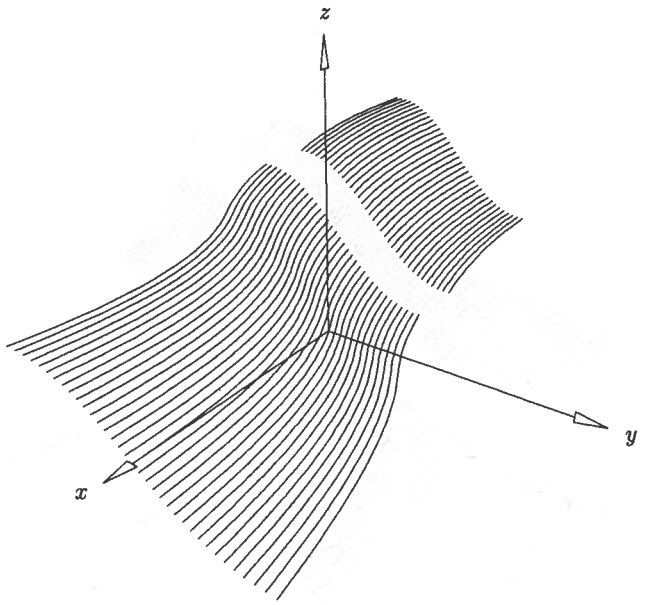


(b)

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.

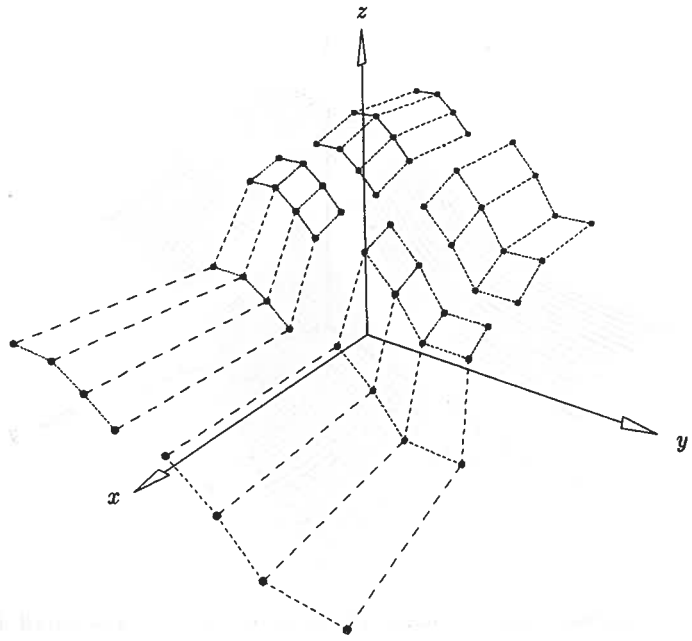


(a)



(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)

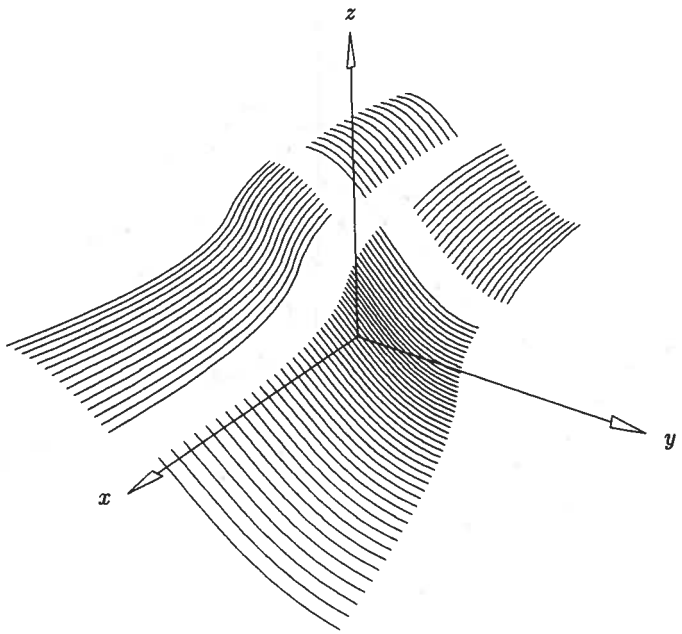
Figure 5.14. Splitting the surface in Figure 5.11 in both u and v directions at $u = 2/5$ and $v = 7/10$. (a) The control net of the split surface; (b) the split surface.

- 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;

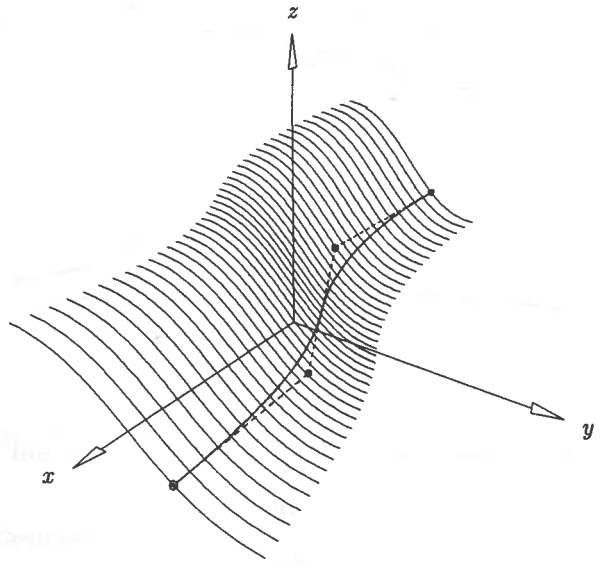


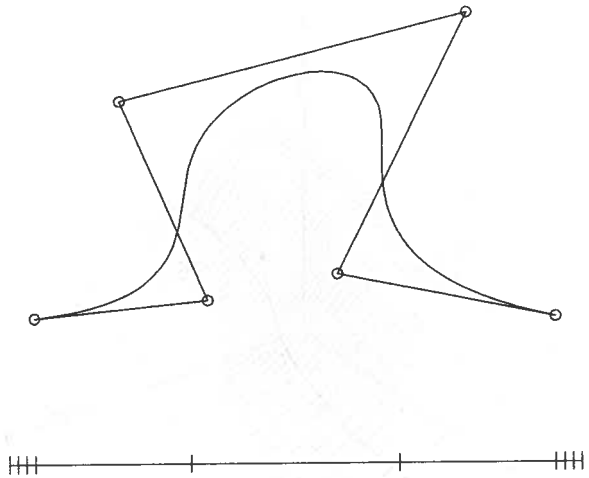
Figure 5.15. Extracting a surface isoparametric curve via knot insertion.

- 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.

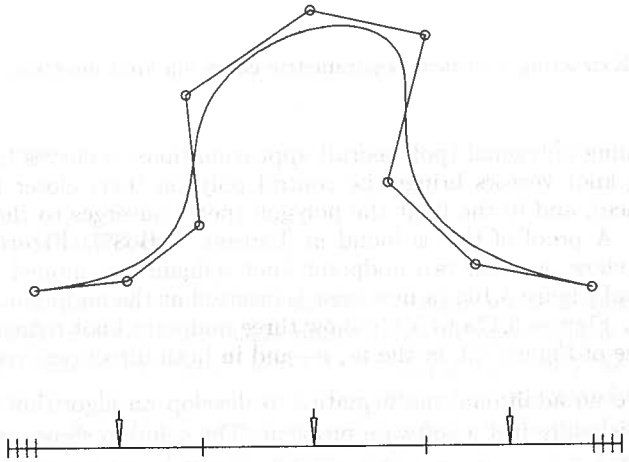
We require no additional mathematics to develop an algorithm for knot refinement; it is really just a software problem. The solution steps are:

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)

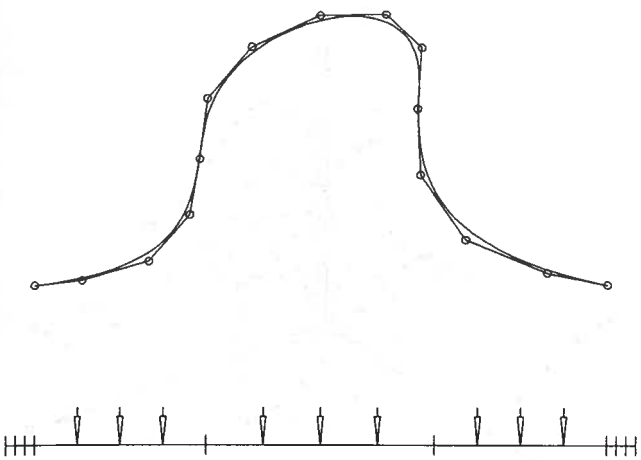


(b)

Figure 5.16. Curve refinement. (a) Cubic curve defined on $\{0,0,0,0,\ 3/10,\ 7/10,1,1,1,1\}$; (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, $\bar{U}$.

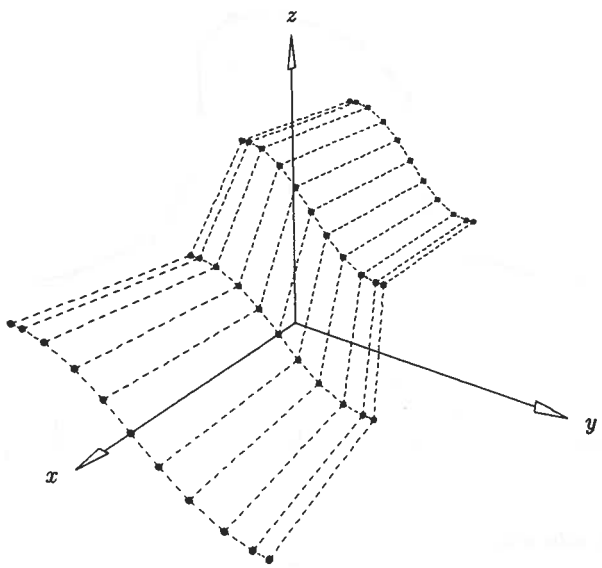
```
ALGORITHM A5.4
  RefineKnotVectCurve(n,p,U,Pw,X,r,Ubar,Qw)
    { /*  Refine curve knot vector  */
      /*  Input:  n,p,U,Pw,X,r  */
```

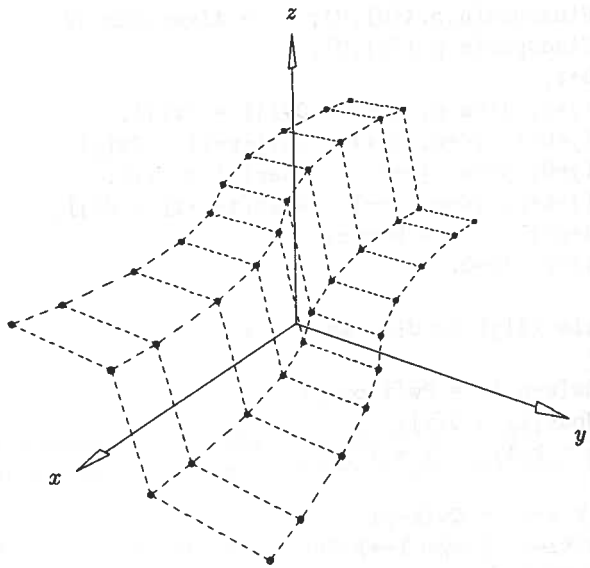
(c)

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
```

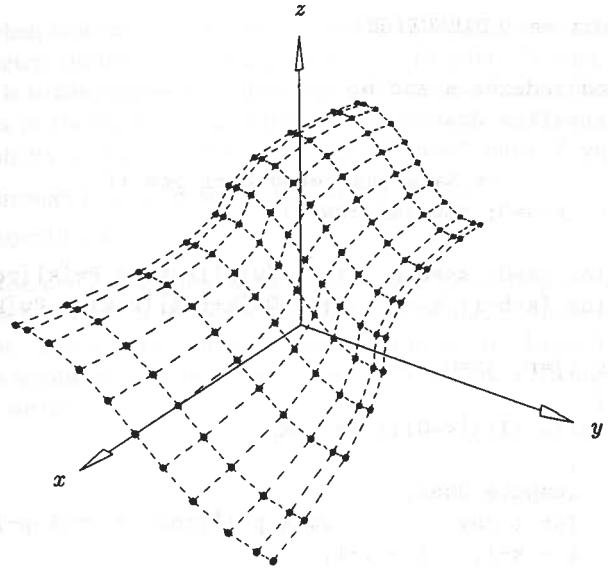


(a)



(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 $\mathbf{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\mathbf{P}^w_{i,j}$ 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.

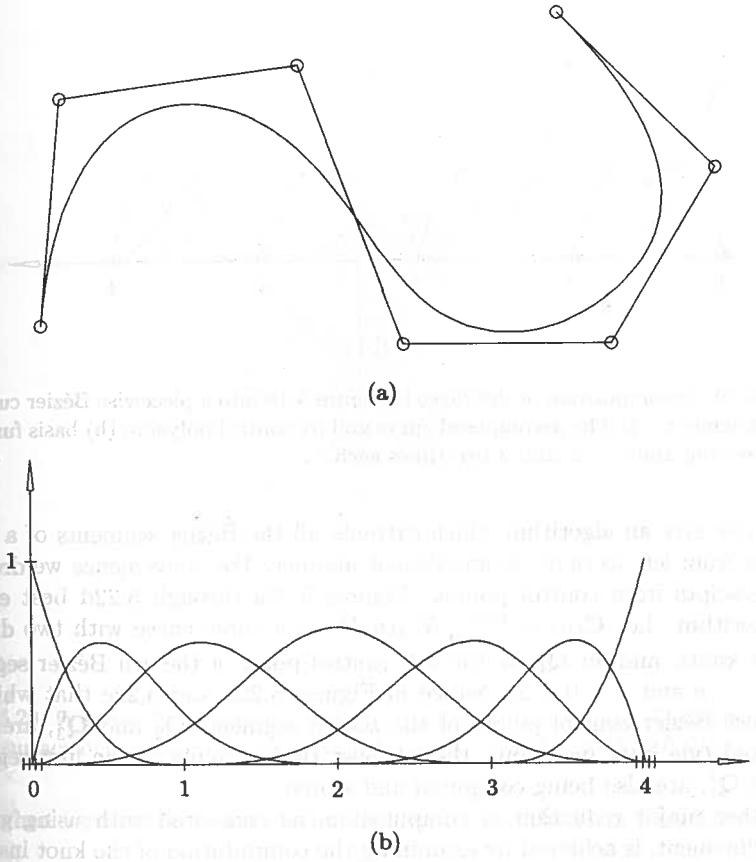
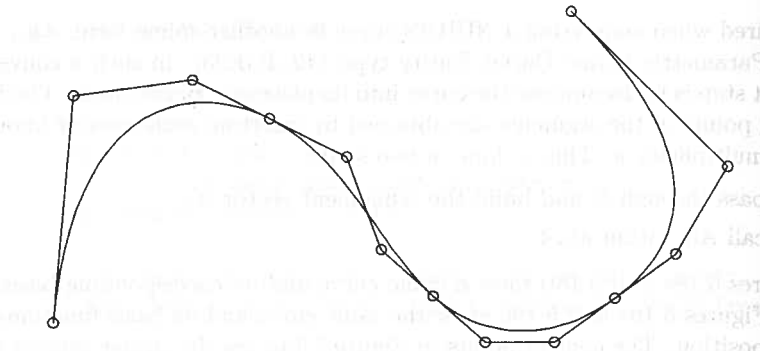
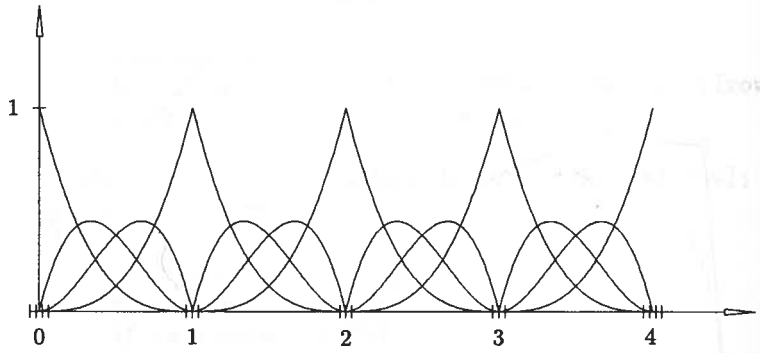


Figure 5.18. A cubic curve. (a) The curve and its control polygon; (b) basis functions defined over $\{0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4\}$.



(a)

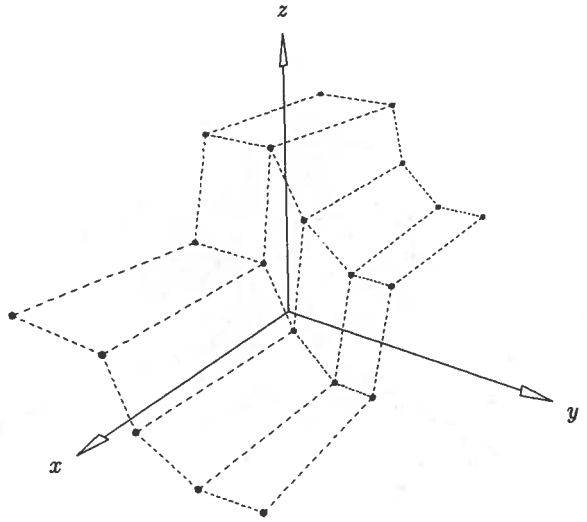


(b)

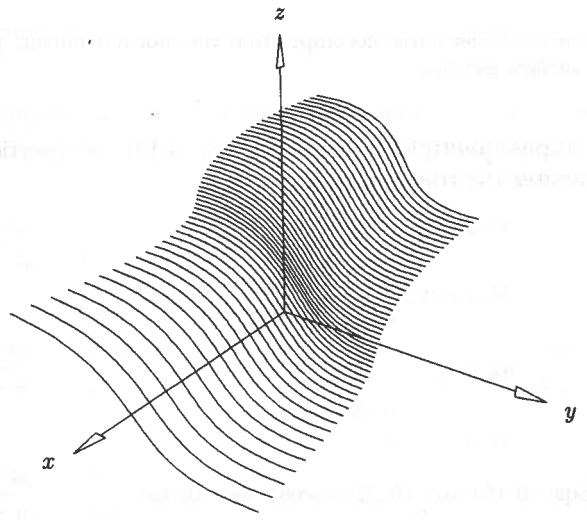
Figure 5.19. Decomposition of the curve in Figure 5.18 into a piecewise Bézier curve via knot refinement. (a) The decomposed curve and its control polygon; (b) basis functions after inserting knots 1, 2, and 3 two times each.

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)

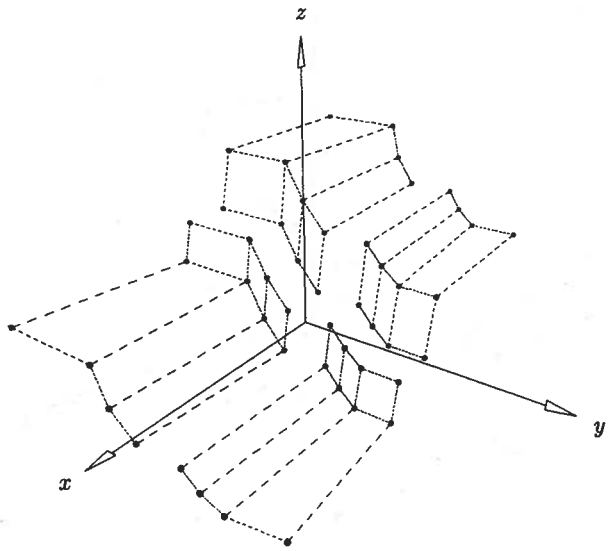


(b)

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

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

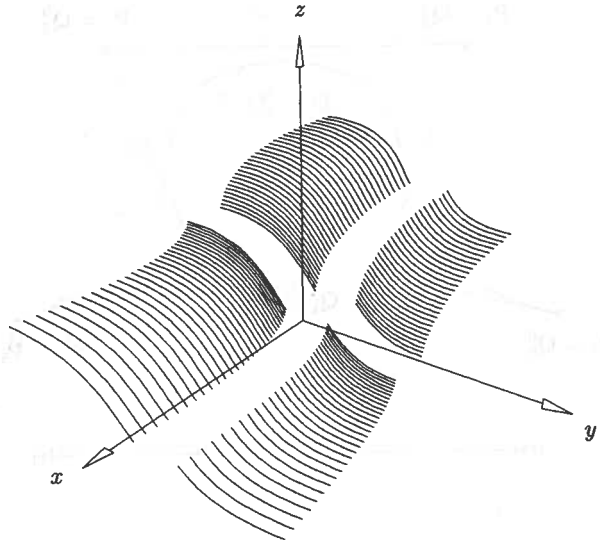
$$\begin{array}{ccccc}
\alpha_{b-p+1,1} & & & & \\
& \alpha_{b-p+2,2} & & & \\
\alpha_{b-p+2,1} & & & & \\
\vdots & & \cdots & \alpha_{b-s,p-s} & \\
\alpha_{b-s-1,1} & & & & \\
& \alpha_{b-s,2} & & & \\
\alpha_{b-s,1} & & & &
\end{array}$$

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

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

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  */
```

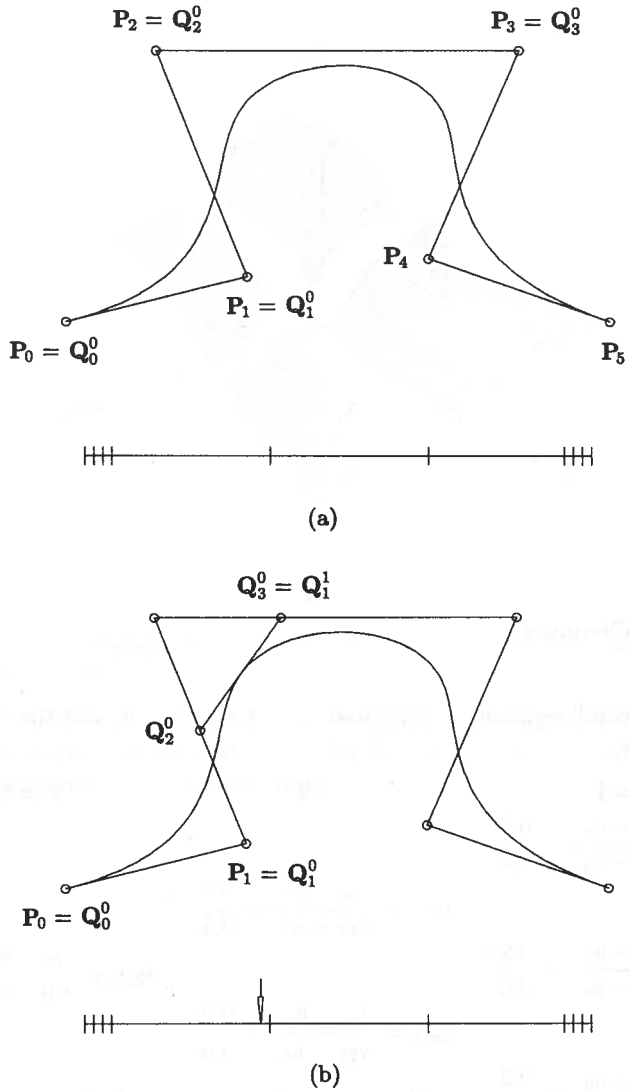


Figure 5.22. On-the-fly curve decomposition. (a) The original cubic curve; (b) first knot inserted; (c) second knot inserted; (d) preparation for the second segment.

```
    /*  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];
```

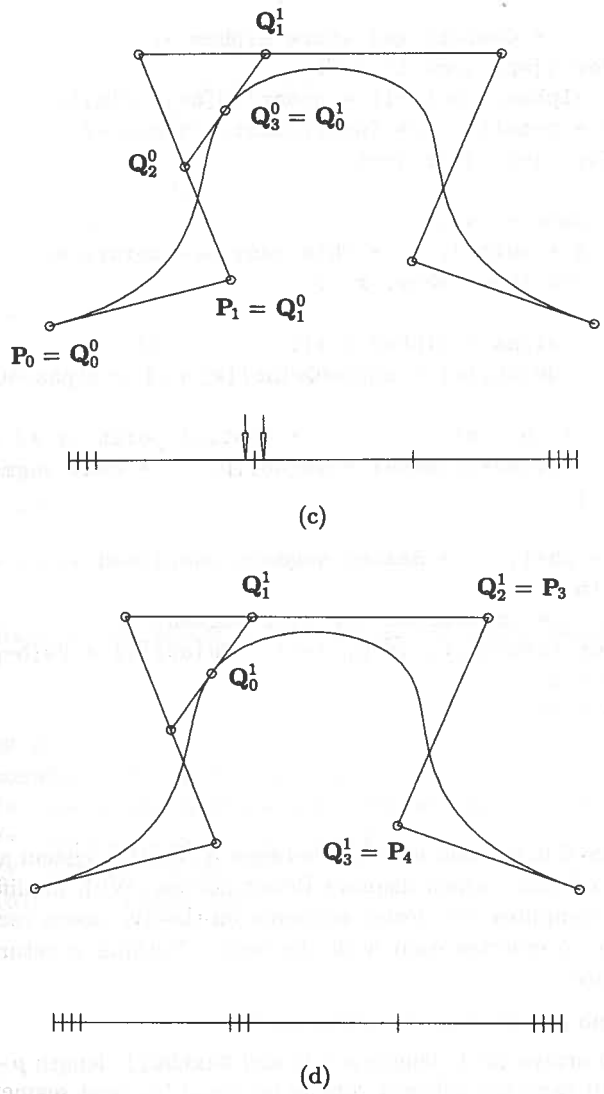



Figure 5.22. (*Continued.*)

```
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.

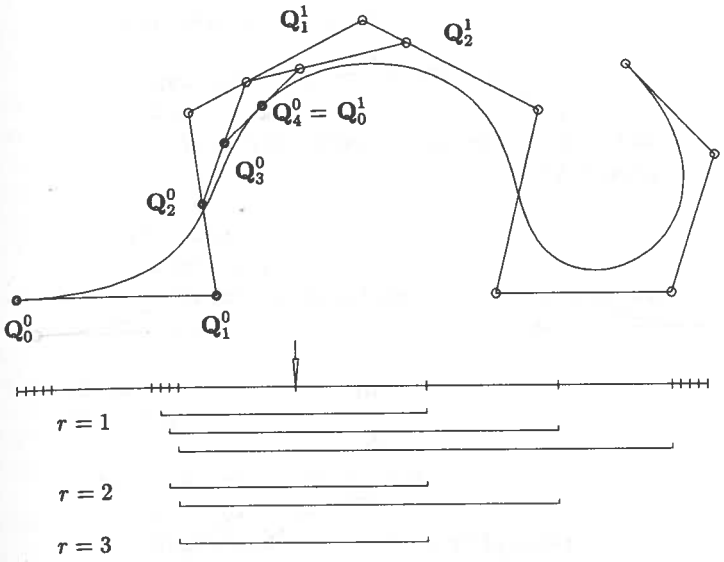
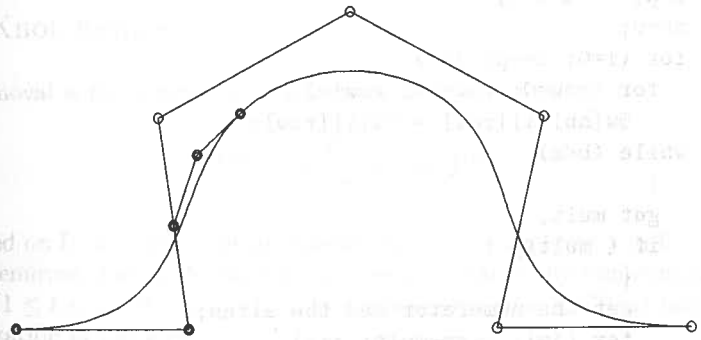


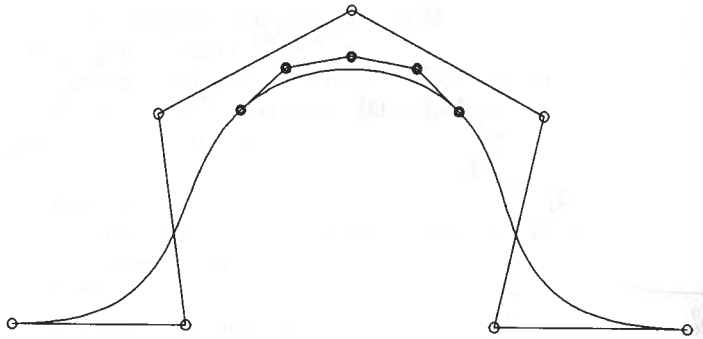
Figure 5.23. Ratios for the alphas used to compute the second Bézier segment of a quartic curve.

```
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)
```

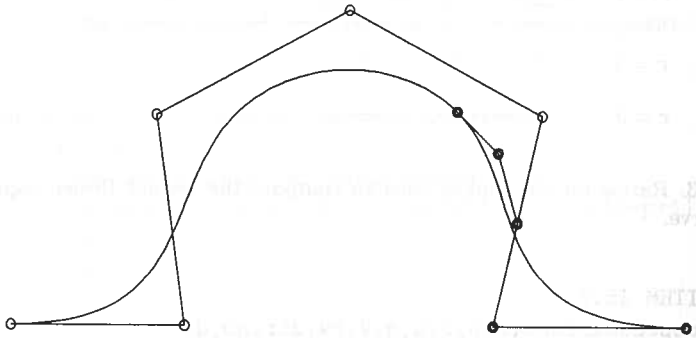


(a)

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

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

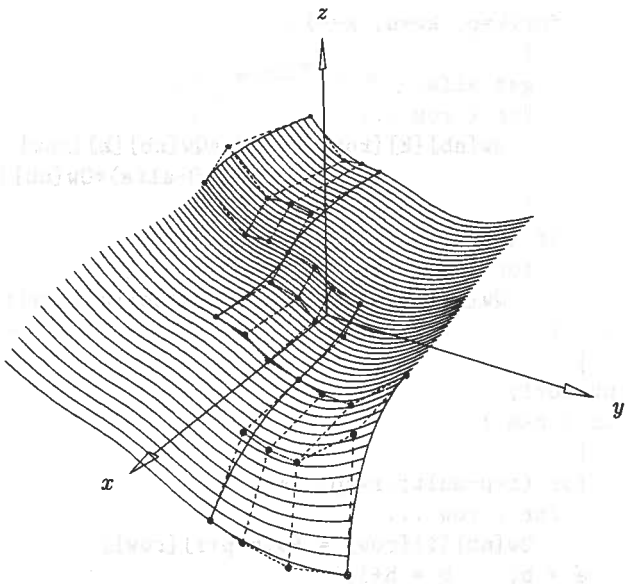


Figure 5.25. Surface decomposition on the fly; three Bézier patches are shown.

where $\tilde{N}_{i,p}(u)$ 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 $u = 1 (= u_6)$.

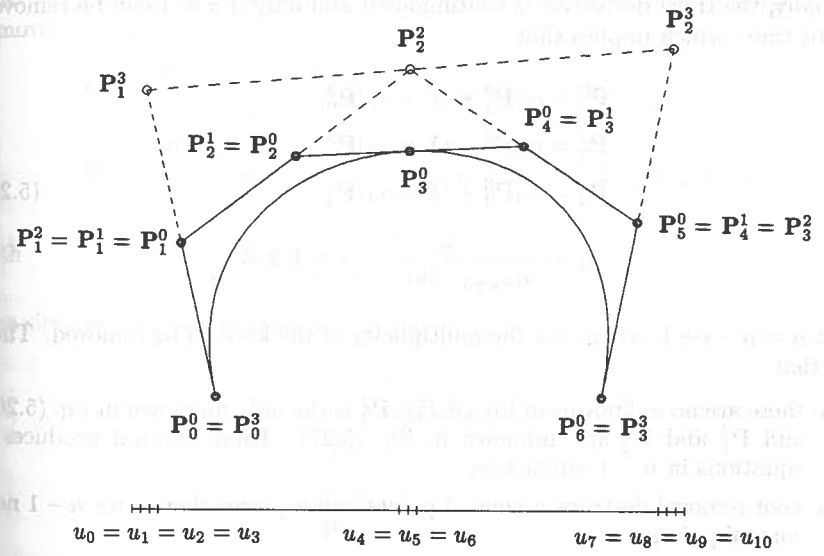


Figure 5.26. Knot removal from a cubic curve with a triple knot.

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

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

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

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

```
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.

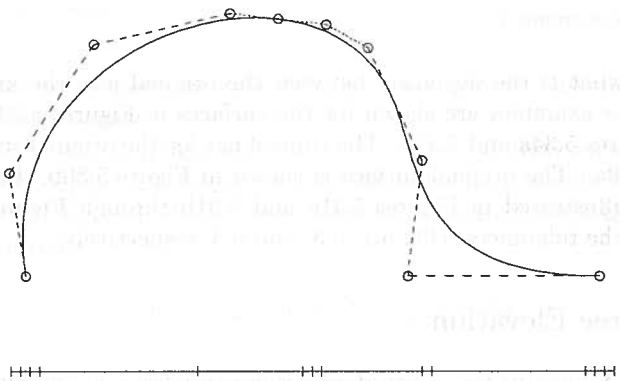
$\mathbf{P}_0^0$	$\mathbf{P}_1^0$	$\mathbf{P}_2^0$	$\mathbf{P}_3^0$	$\mathbf{P}_4^0$	$\mathbf{P}_5^0$	$\mathbf{P}_6^0$	before loop
			X				$t=0$
		$\mathbf{P}_2^2$	X	$\mathbf{P}_2^2$			$t=1$
	$\mathbf{P}_1^3$	X	X	X	$\mathbf{P}_2^3$		$t=2$

- 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.

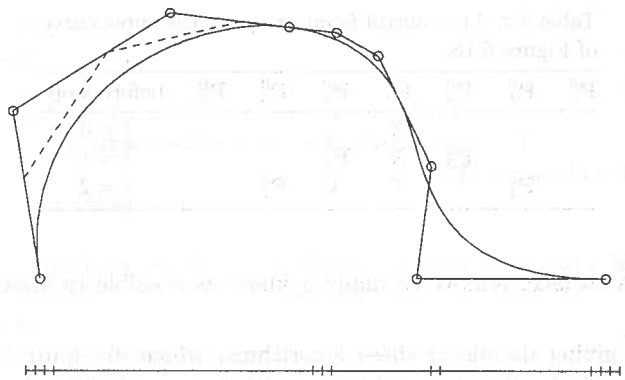
Surface knot removal examples are shown in Figures 5.28a–5.28d. Figure 5.28a shows the original surface. In Figure 5.28b the triple knot, $u = 3/10$, is removed three times. In Figure 5.28c the single knot, $v = 1/4$, is removed, and in Figure 5.28d both knots are removed at the same time. In Figure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

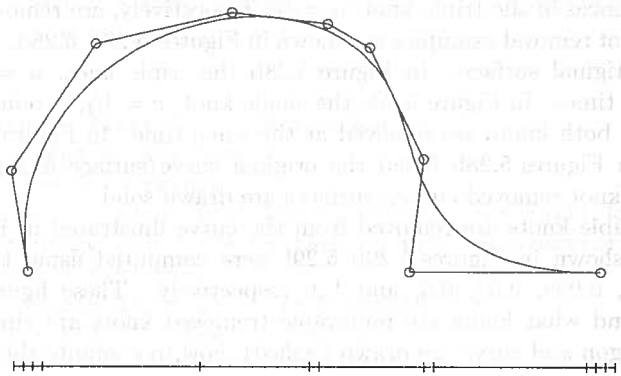


(a)

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)

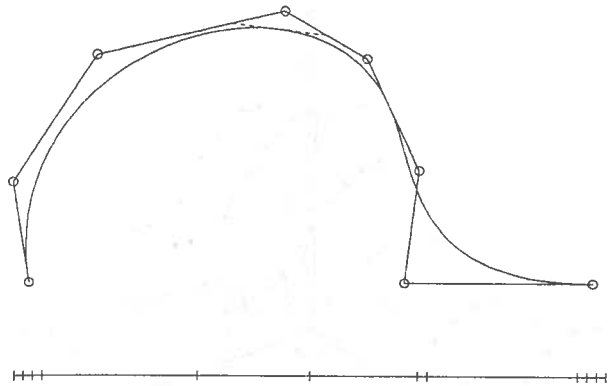
Figure 5.27. (*Continued.*)

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.

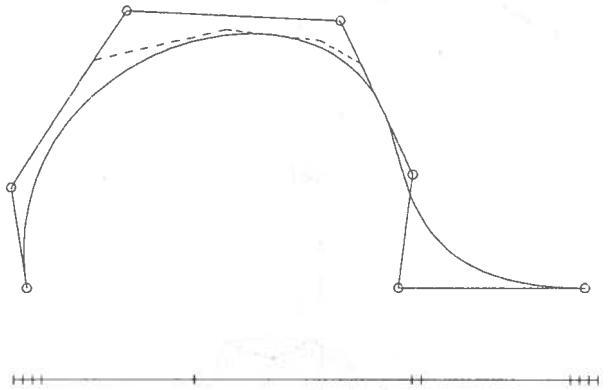
5.5 Degree Elevation

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.*)

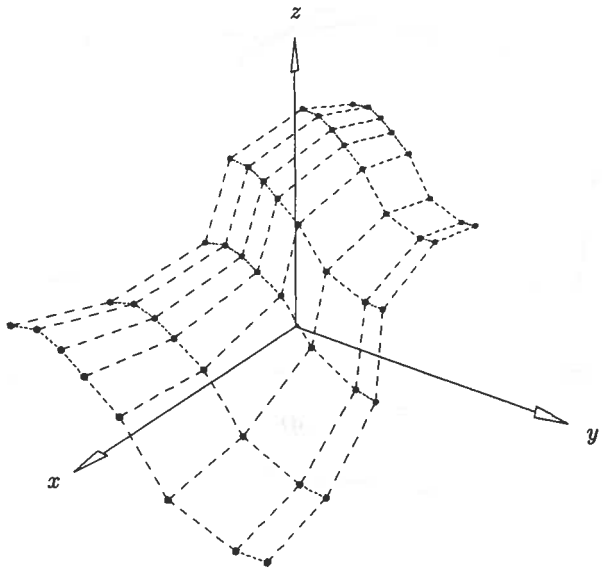
From a vector space point of view, $\mathbf{C}_{n+1}(u)$ is simply $\mathbf{C}_n(u)$ embedded in a higher dimensional space.

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

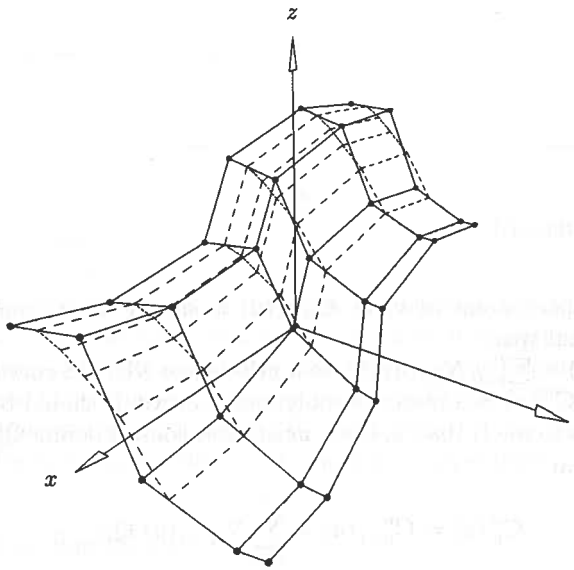
$$\mathbf{C}_p^w(u) = \mathbf{C}_{p+1}^w(u) = \sum_{i=0}^{\hat{n}} N_{i,p+1}(u)\ \mathbf{Q}_i^w \tag{5.31}$$

$\mathbf{C}_{p+1}^w(u)$ 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:

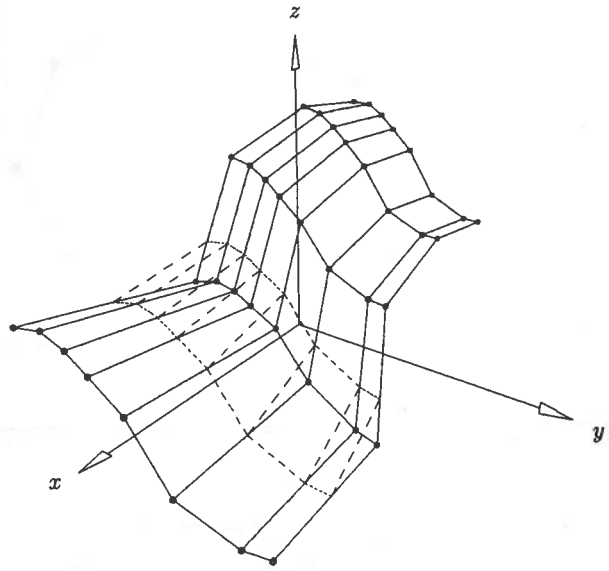


(a)

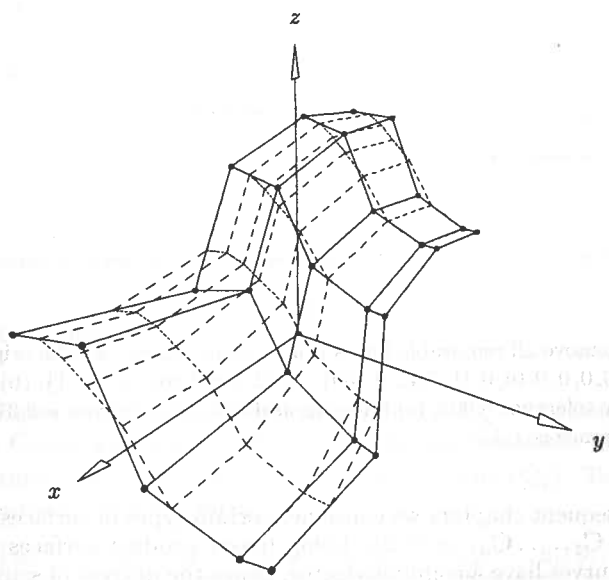


(b)

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.

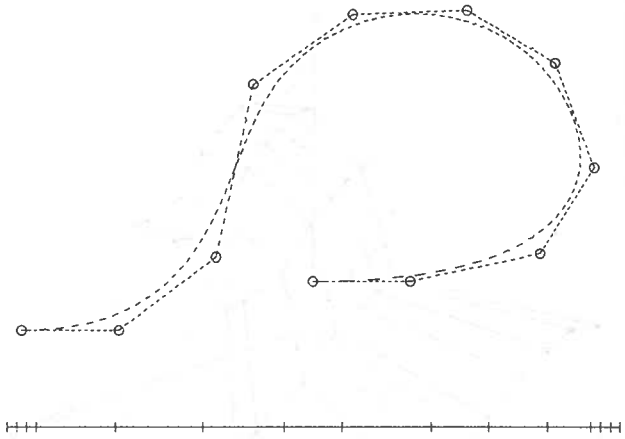


(c)

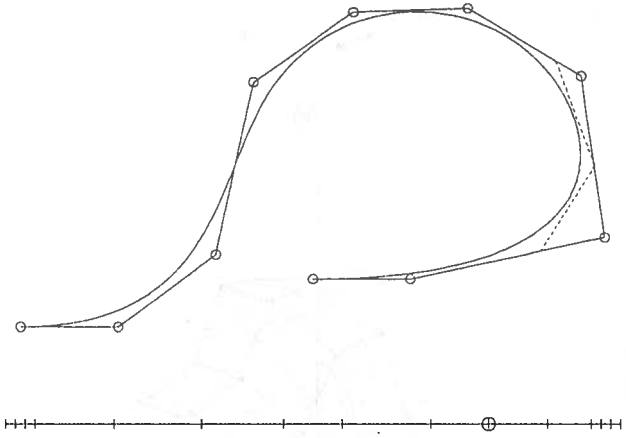


(d)

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.



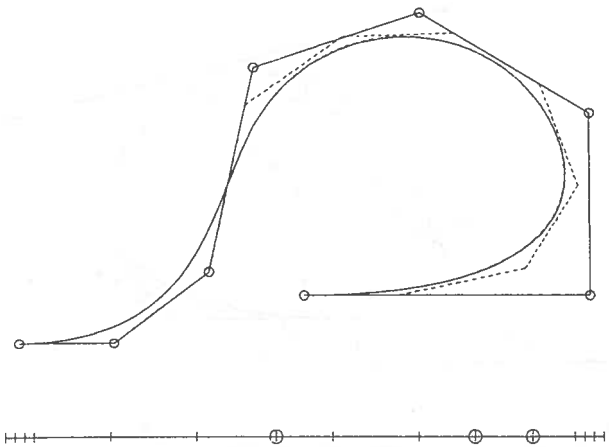
(a)



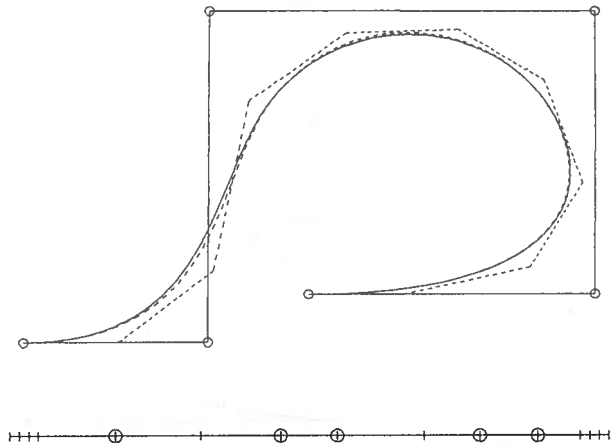
(b)

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)

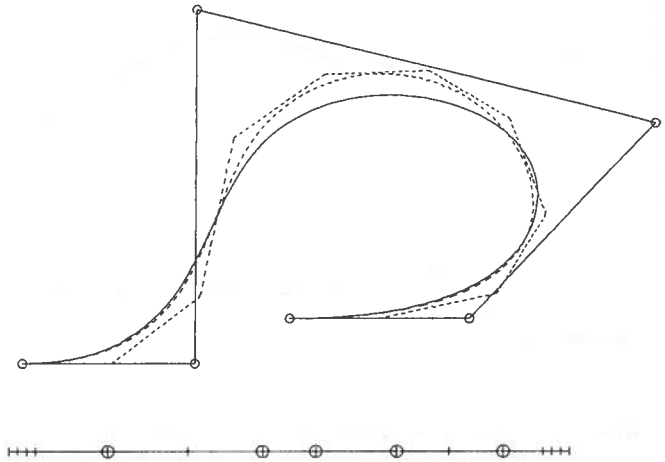
Figure 5.29. (*Continued.*)

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.

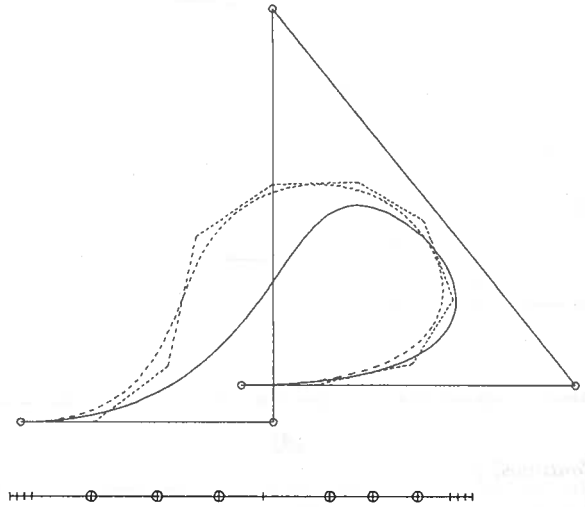
There are three unknowns in Eq. (5.31), $\hat{n}$, $\hat{U}$, and the $\{\mathbf{Q}_i\}$. To determine $\hat{n}$ and $\hat{U}$ assume that U has the form

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

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)

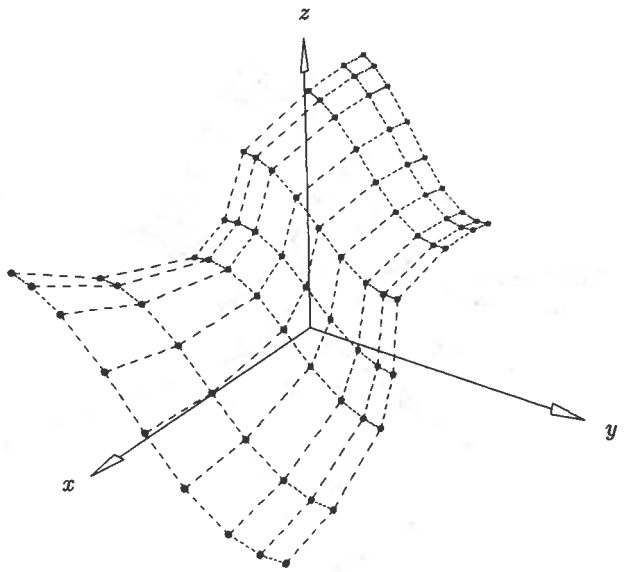
Figure 5.29. (*Continued.*)

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

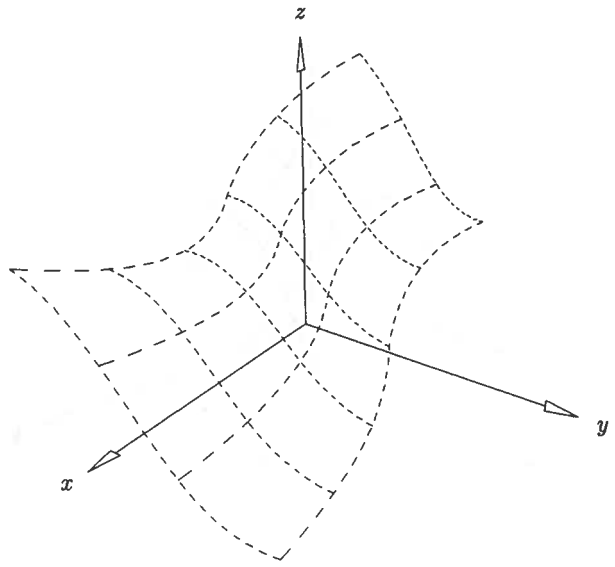
$$\hat{n} = n + s + 1 \tag{5.32}$$

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

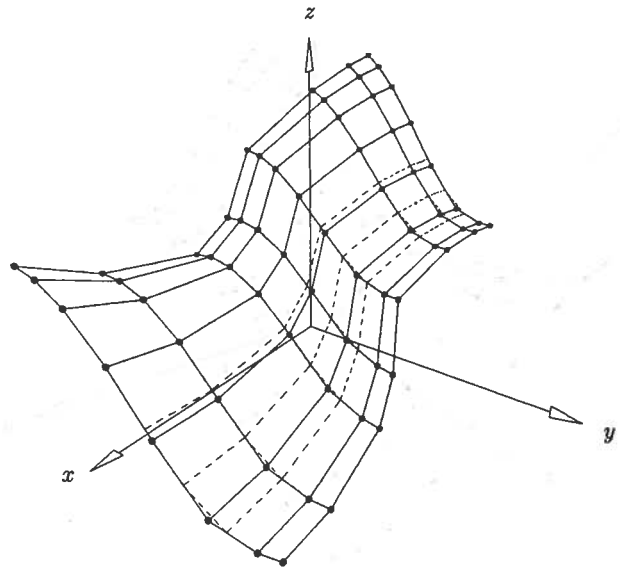


(a)

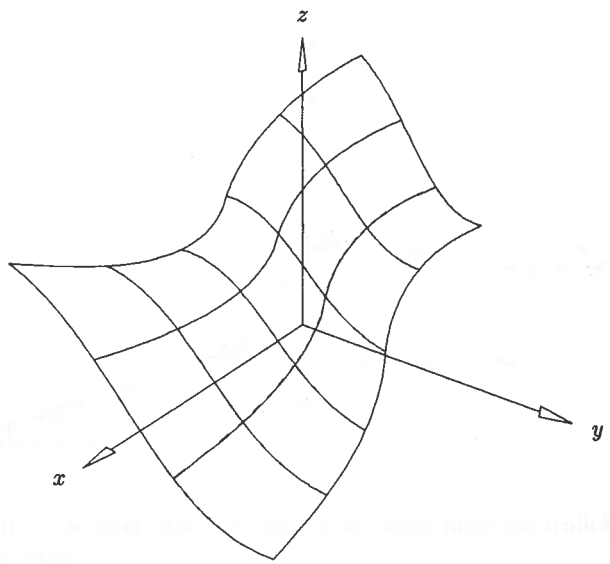


(b)

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 $V = \{0, 0, 0, 0.2, 0.42, 0.5, 0.81, 0.9, 1, 1, 1\}$.

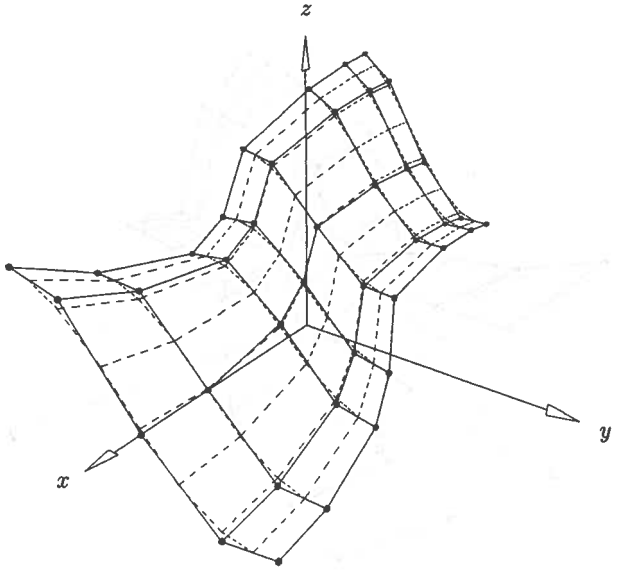


(a)

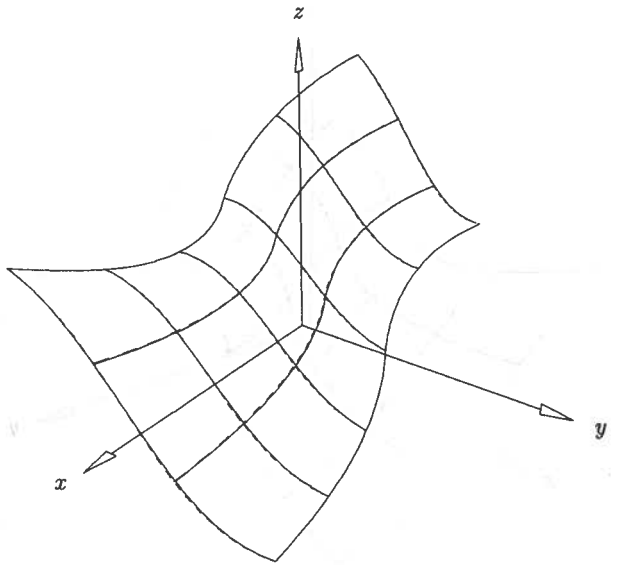


(b)

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.

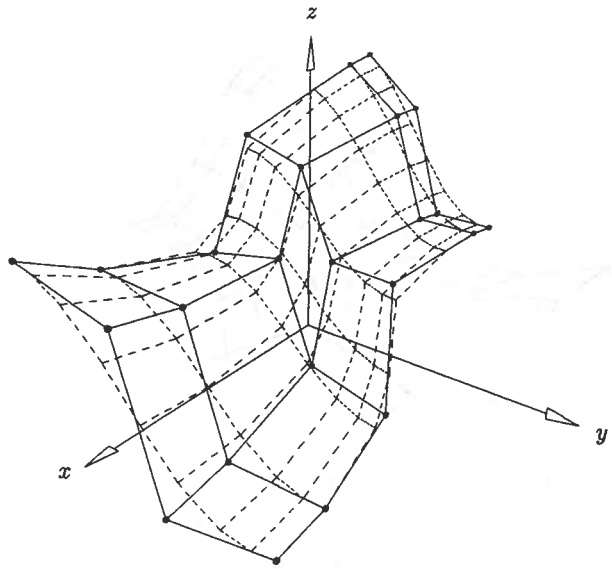


(a)

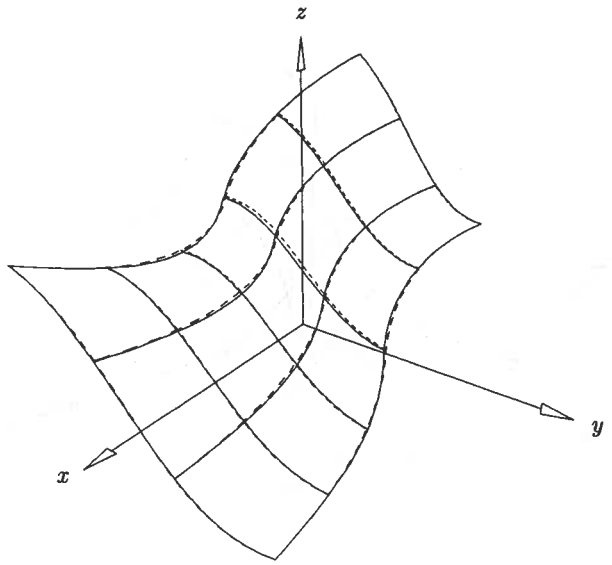


(b)

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.

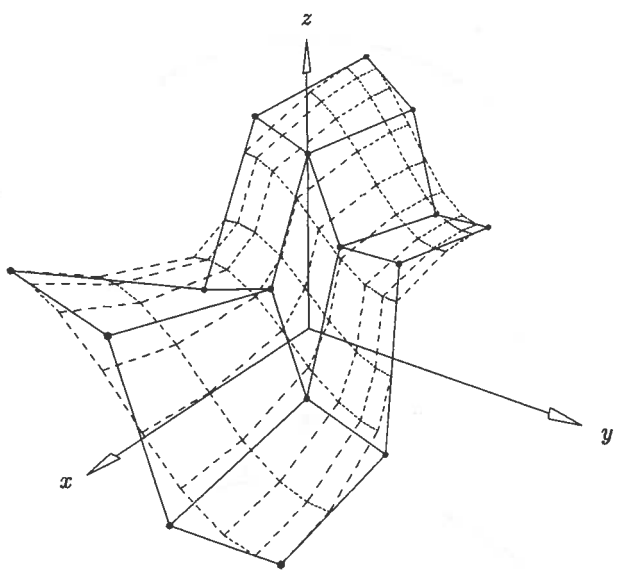


(a)

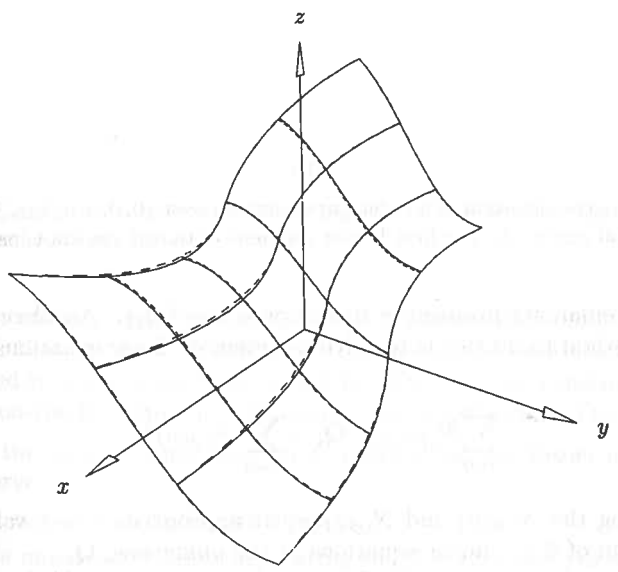


(b)

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.

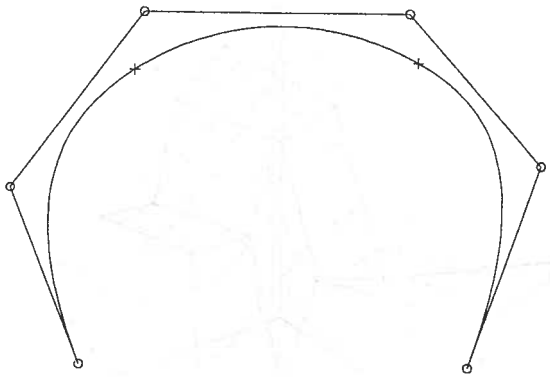


(a)

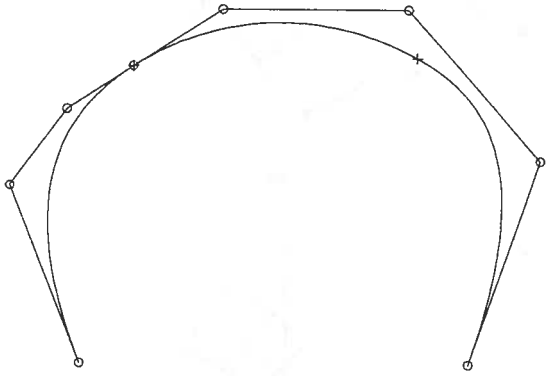


(b)

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)

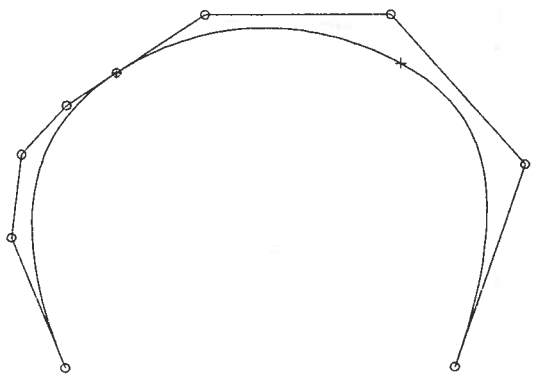
Figure 5.35. Degree elevation of a cubic curve defined over $\{0,0,0,0,\,3/10,\,7/10,1,1,1,1\}$. (a) The original curve; (b) the first Bézier segment obtained via knot insertion.

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

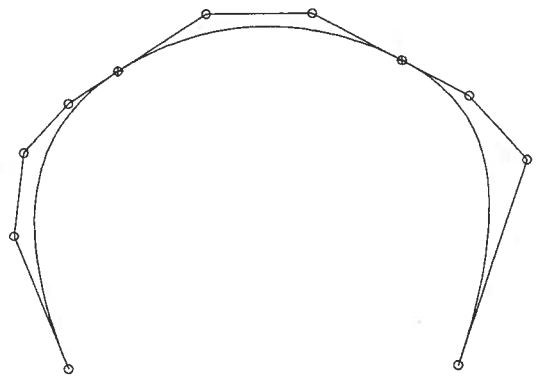
$$\sum_{i=0}^{\hat{n}} N_{i,p+1}(u)\,\mathbf{Q}_i = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i$$

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 ($p \to p+1$).



(c)



(d)

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

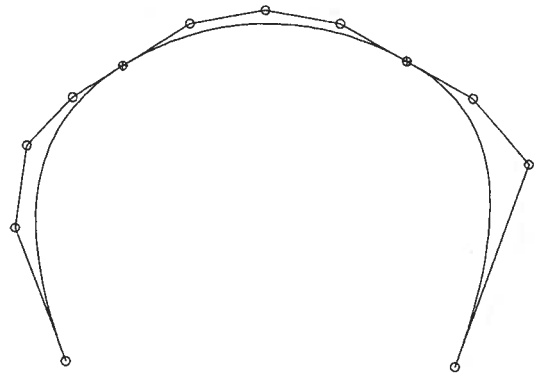
We present another algorithm here which is mathematically simpler, and competitive with that given in [Prau91], particularly in the case where the degree is to be raised by more than 1 (see [Pieg94]). The solution consists of applying three steps, on-the-fly, while moving through the knot vector. The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

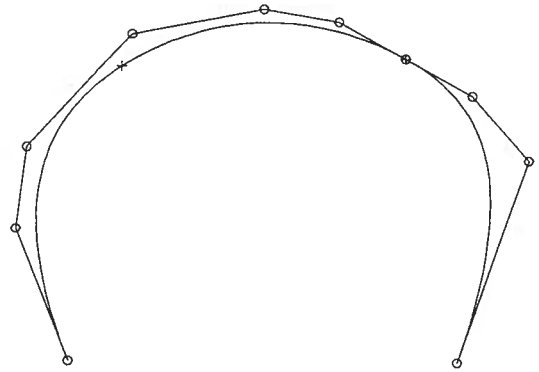
Figures 5.35a–5.35h show the progress of the algorithm. The original curve is a cubic with knot vector $U = \{0,0,0,0,u_a,u_b,1,1,1,1\}$. It is elevated to fourth-degree. The figures are explained as:

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;

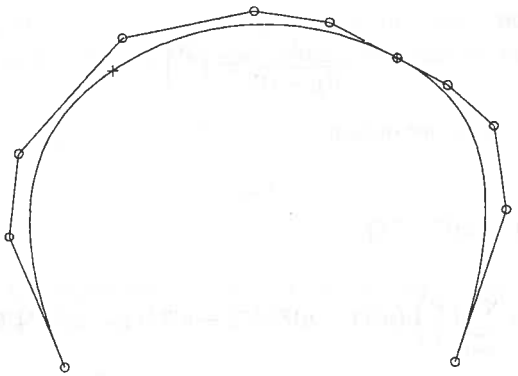
f. two occurrences of the knot u_a are removed;

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

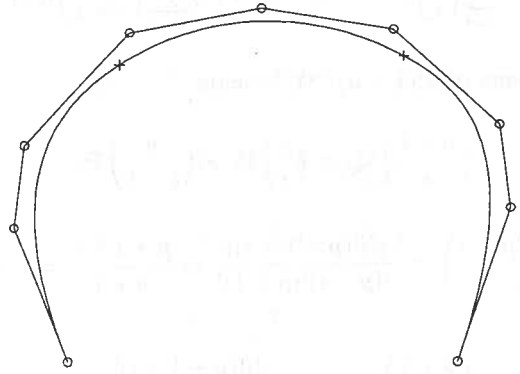
h. two occurrences of u_b are removed, yielding the final fourth-degree, C^2 continuous curve.

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. Moreover, the α_js of Eq. (5.29) do not change, regardless of the degree (but the α_is can, in general, change). This is due to the fact that the algorithm has already inserted u_e to have multiplicity $p+1$. Hence, 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;
   /* Compute Bézier degree elevation coefficients */
  ph2 = ph/2;
  bezalfs[0][0] = bezalfs[ph][p] = 1.0;
  for (i=1; i<=ph2; i++)
```

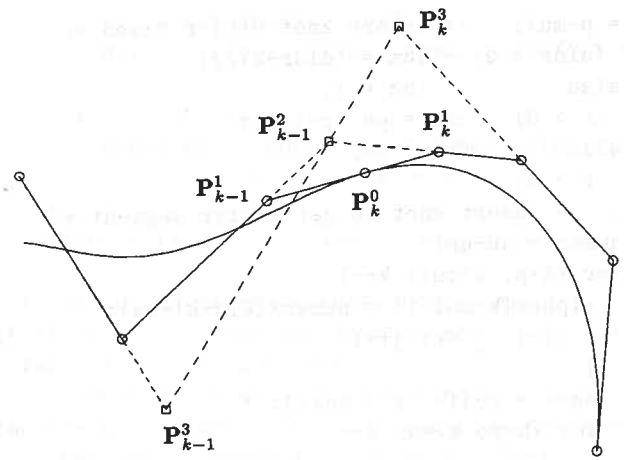



Figure 5.36. Removal of three knots from a quartic curve.

```
  {
  inv = 1.0/Bin(ph,i);
  mpi = Min(p,i);
  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;
ua = U[0];
for (i=0; i<=ph; i++)   Uh[i] = ua;
kind = ph+1;
Qw[0] = Pw[0];   cind = 1;
lbz = 1;   r = -1;   /* Initialize first Bézier seg */
for (i=0; i<=p; i++)   bpts[i] = Pw[i];
a = p;    b = p+1;
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--)
    alphas[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--)
      {
      alf = alphas[k-s];
      bpts[k] = alf*bpts[k]+(1.0-alf)*bpts[k-1];
      }
    Nextbpts[save] = bpts[p];
    }
  }  /* End of "insert knot" */
if (r > 0)  RBZ = rbz+1;  /* Degree elevate Bézier */
  else      RBZ = rbz;
for (i=lbz; i<=RBZ; i++)
  {  /* Only points lbz,...,RBZ 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 */
  if (oldr > 2)
    {  /* Alphas on the right do not change */
    alfj = (ua-Uh[kind-1])/(ub-Uh[kind-1]);
    }
  first = kind-2;   last = kind;
  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)
        {
```

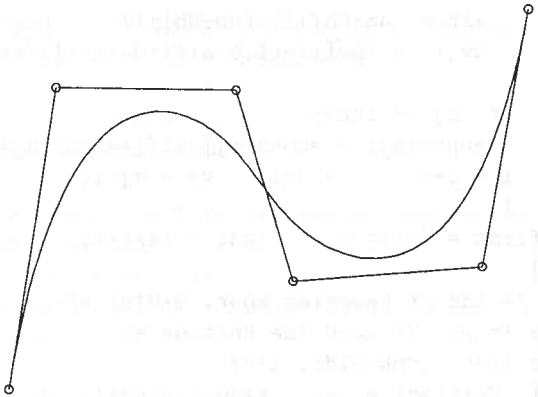


```
        alf = (ua-Uh[i])/(ub-Uh[i]);
        Qw[i] = (Qw[i]-(1.0-alf)*Qw[i-1])/alf;
        }
      if (kj >= lbz)
        ebpts[kj] = ebpts[kj]-alfj*ebpts[kj+1]/(1.0-alfj);
      i = i+1;    j = j-1;    kj = kj-1;
      }
    first = first-1;     last = last+1;
    }
  }  /* End of removing knot, 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 */
  lbz = 1;
  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;
}
```

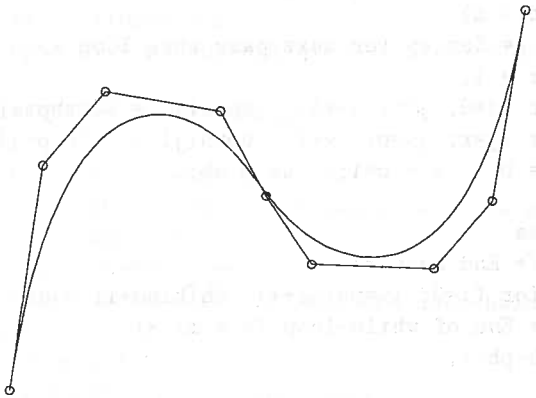
Several curve degree elevation examples are shown in Figures 5.37a–5.37d. The original third-degree curve (Figure 5.37a) is raised to fourth-, fifth-, and seventh-degrees in Figures 5.37b, 5.37c, and 5.37d, respectively. Note that the control polygon converges to the curve as the degree is raised.

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  */
```



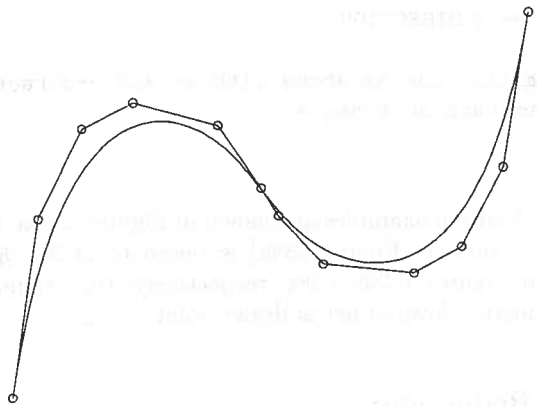
(a)



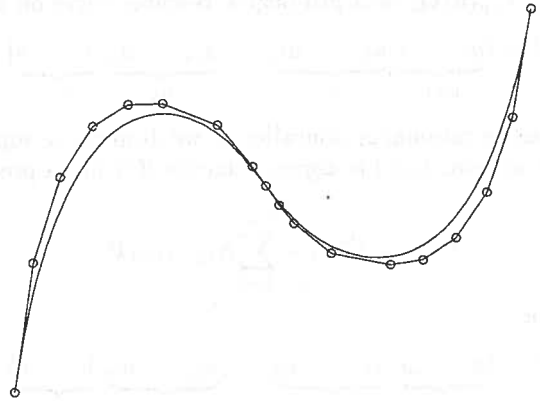
(b)

Figure 5.37. Curve degree elevation example. (a) The original cubic curve defined over $\{0,0,0,0,\frac{4}{10},\frac{7}{10},1,1,1,1\}$; (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

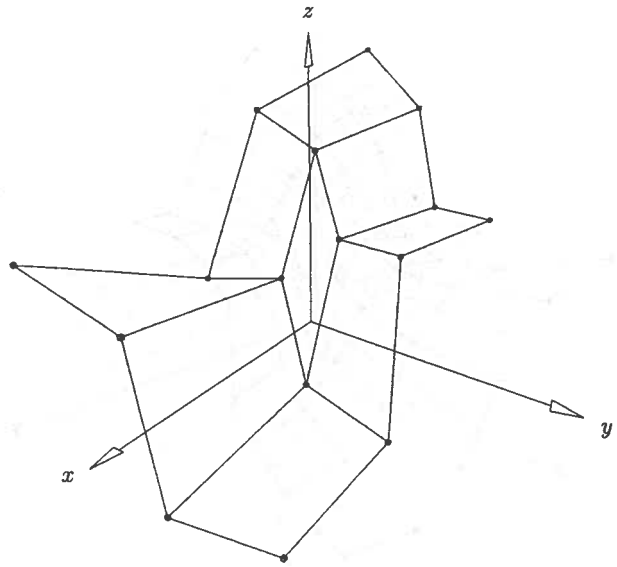
$$\hat{U} = \{\underbrace{a,\ldots,a}_{p}, \underbrace{u_1,\ldots,u_1}_{m_1-1}, \ldots, \underbrace{u_s,\ldots,u_s}_{m_s-1}, \underbrace{b,\ldots,b}_{p}\} \tag{5.38}$$

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

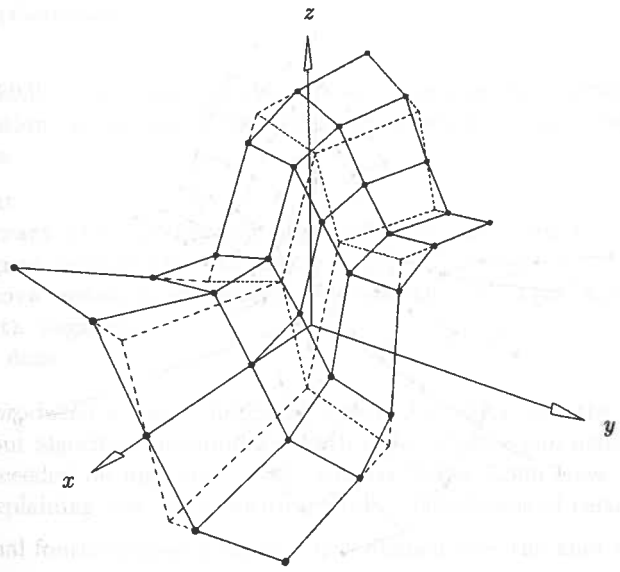
Note that m_i can be 1, which implies that the knot u_i is not present in $\hat{U}$. This means that u_i was precisely removable from $\mathbf{C}(u)$. Although it is always possible to degree elevate a curve, clearly a curve may not be degree reducible. The situation is similar to knot removal in that 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 $E(u) = |\,\mathbf{C}(u) - \hat{\mathbf{C}}(u)\,|$ 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

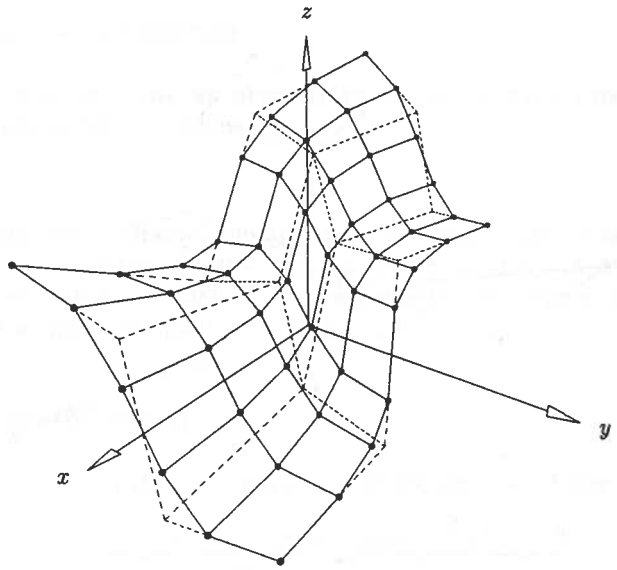


(a)

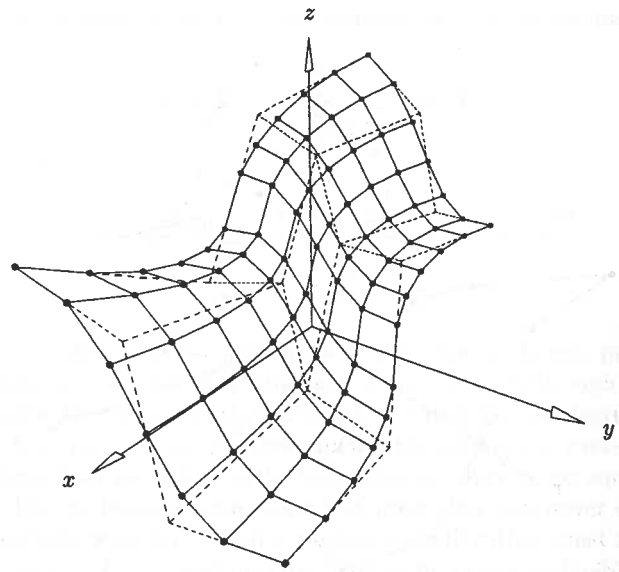


(b)

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)

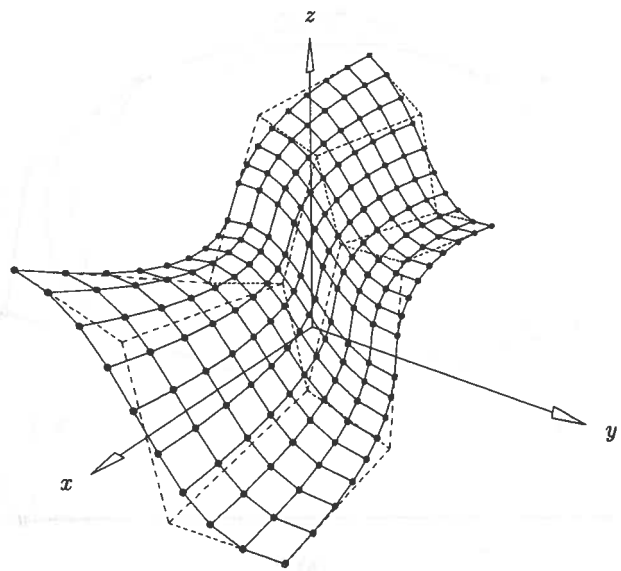


(d)

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

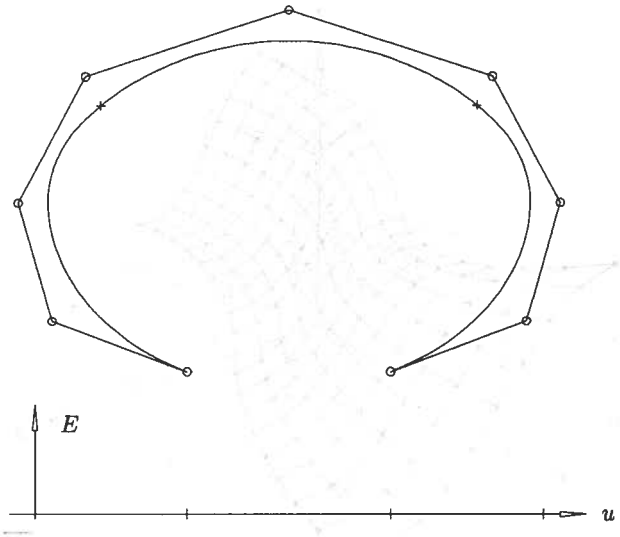
Error is produced in both the Bézier degree reduction and the knot removal steps, and our algorithm accumulates both types of error and exits immediately if `TOL` is exceeded on any knot span. Figures 5.39a–5.39h show walk-through examples explaining how the algorithm works. The details of each step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

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

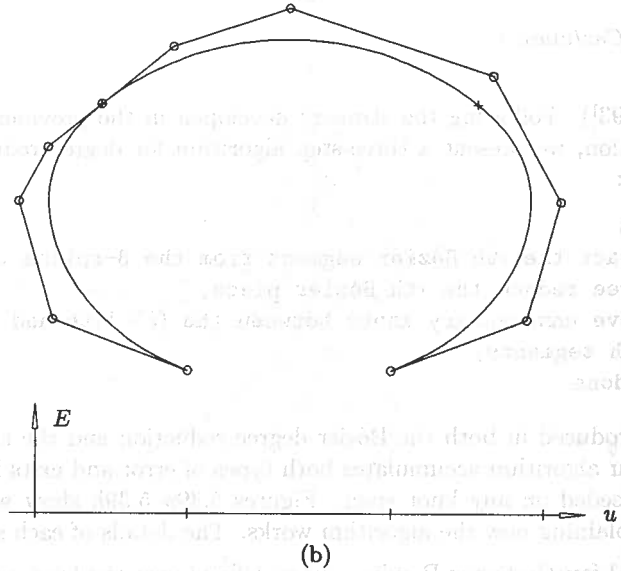
$$U = \{0, 0, 0, 0, 0, u_1, u_1, u_2, u_2, 1, 1, 1, 1, 1\}$$

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;



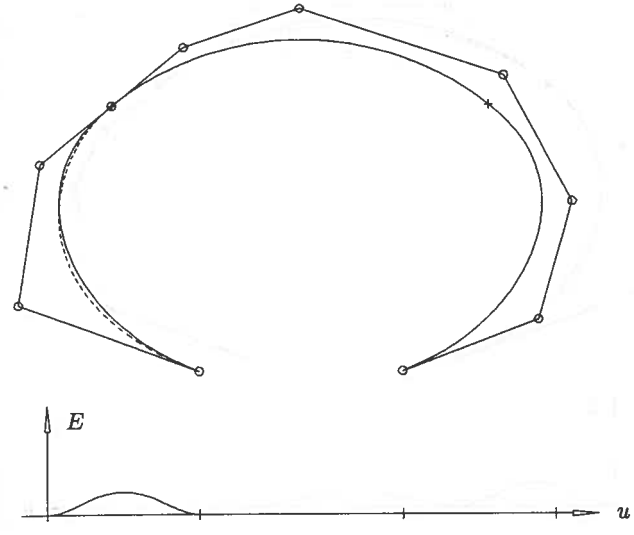
(a)



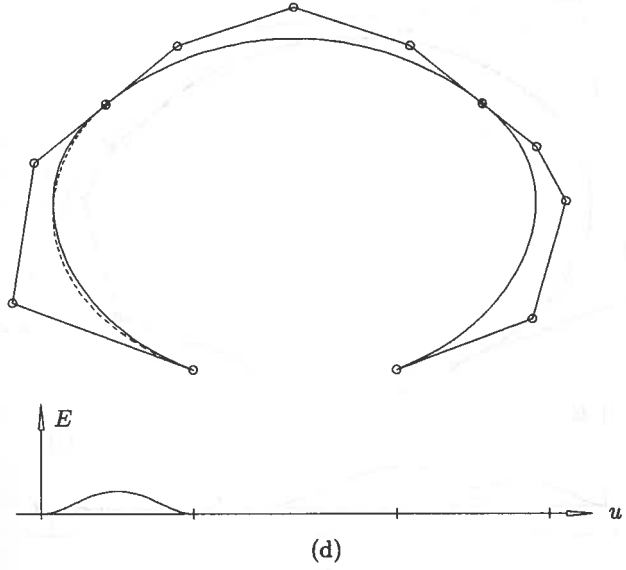
(b)

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)



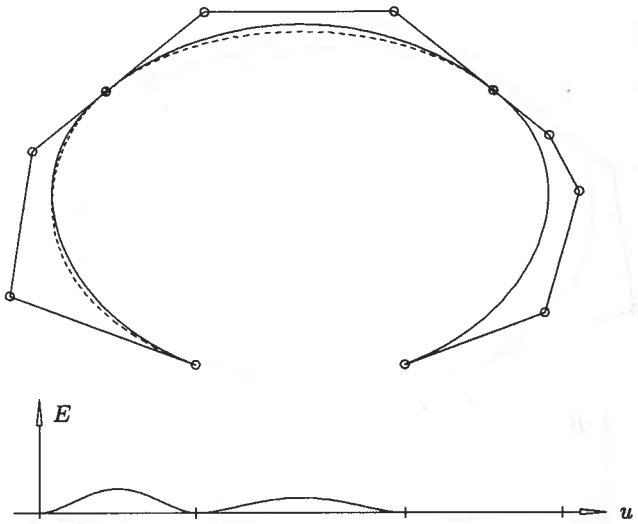
(d)

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

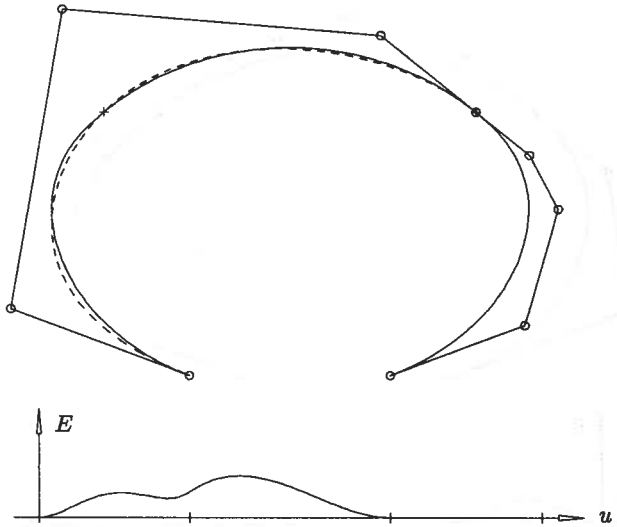
d. the knot u_2 is inserted twice to obtain the second Bézier segment;

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)



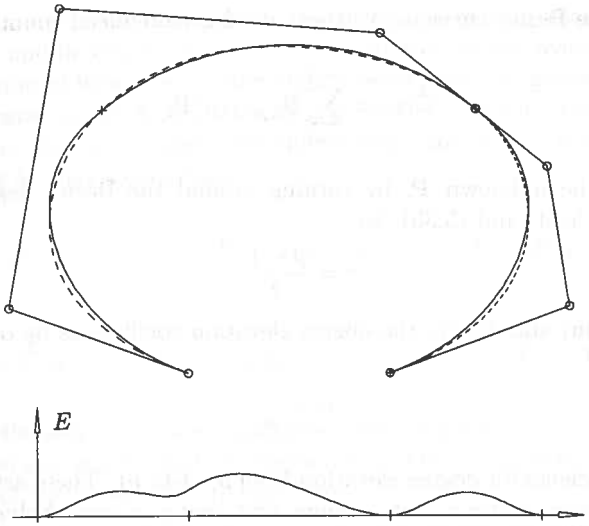
(f)

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

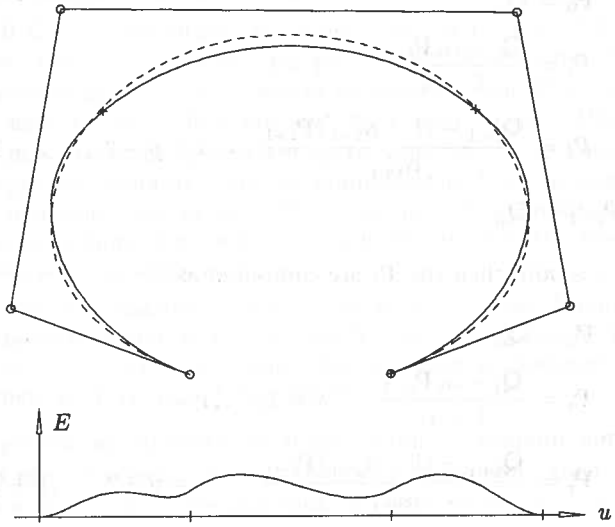
g. the last Bézier segment is degree reduced;

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

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

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:

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

p odd:

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

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

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

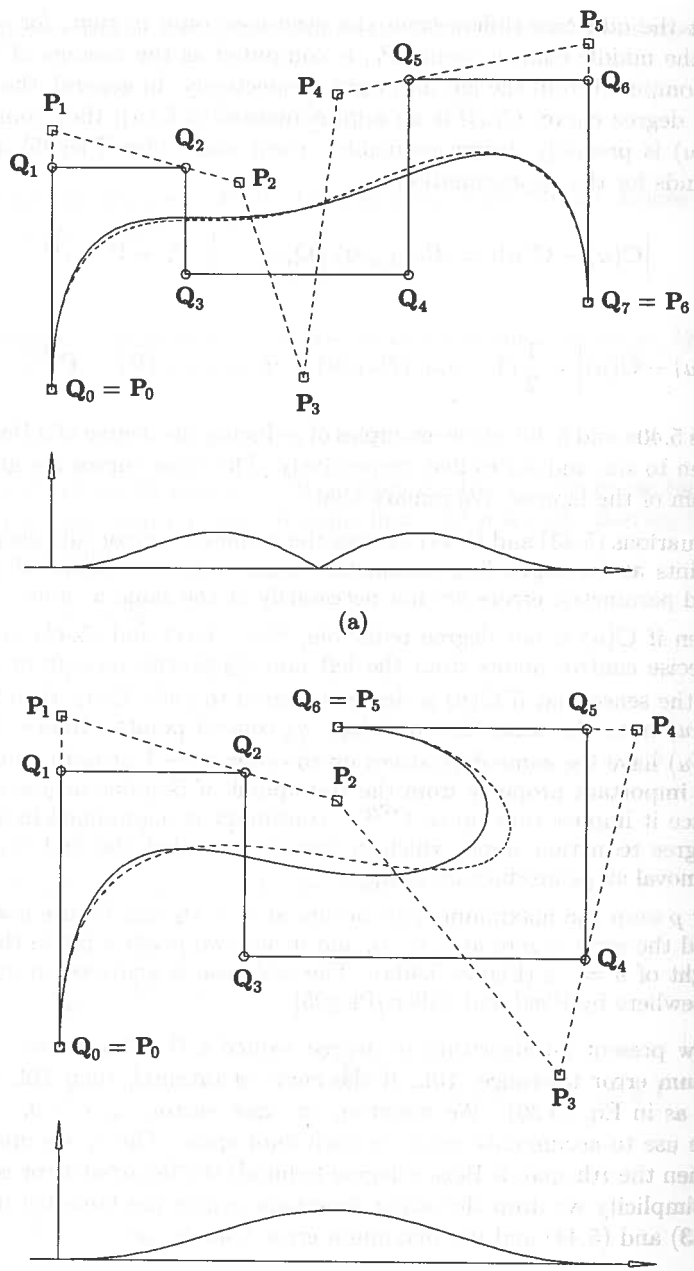


Figure 5.40. Bézier degree reduction. (a) From degree seven to degree six; (b) from degree six to degree five.

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

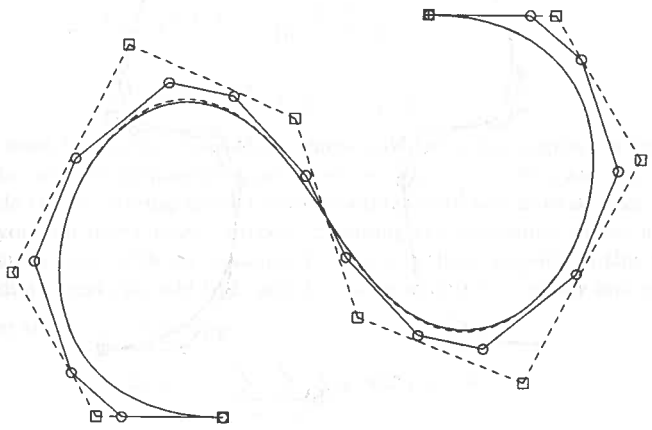
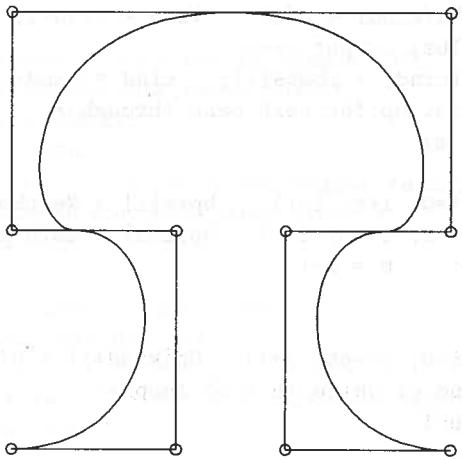


Figure 5.41. Degree reduction of a B-spline curve from degree five to four.

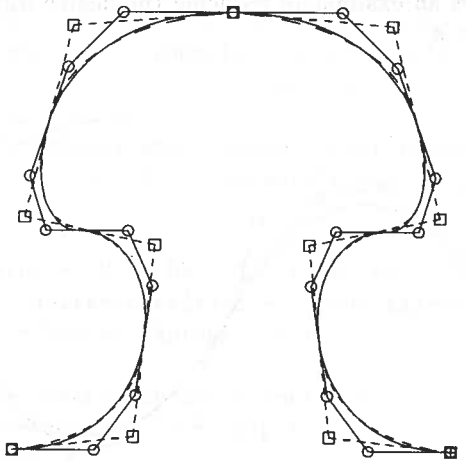
$$U = \{0,0,0,0,0,0,0.15,0.15,0.3,0.3,0.5,0.5,$$
$$0.7,0.7,0.85,0.85,1,1,1,1,1,1\}$$

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