CHAPTER [redacted]

Geometric Representation

One of the major concepts in computer graphics is *modeling* of objects. By this we mean numerical description of the objects in terms of their geometric property (size, shape) and how they interact with light (reflect, transmit). The focus of this chapter is on geometric representation of objects. We will discuss illumination and shading models in subsequent chapters.

A graphics system typically uses a set of primitives or geometric forms that are simple enough to be efficiently implemented on the computer but flexible enough to be easily manipulated (assembled, deformed) to represent or model a variety of objects. Geometric forms that are often used as primitives include, in order of complexity, points, lines, polylines, polygons, and polyhedra. More complex geometric forms include curves, curved surface patches, and quadric surfaces.

9.1 SIMPLE GEOMETRIC FORMS

Points and Lines

Points and lines are the basic building blocks of computer graphics. We specify a point by giving its coordiantes in three- (or two-) dimensional space. A *line* or *line segment* is specified by giving its endpoints $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$.

Polylines

A *polyline* is a chain of connected line segments. It is specified by giving the vertices (nodes) $P_0, \ldots, P_N$ defining the line segments. The first vertex is called the *initial* or *starting point* and the last vertex, the *final* or *terminal point* (see Fig. 9-1).

Polygons

A *polygon* is a closed polyline, that is, one in which the initial and terminal points coincide. A polygon is specified by its *vertex list* $P_0, \ldots, P_N, P_0$. The line segments $\overline{P_0P_1}, \overline{P_1P_2}, \ldots, \overline{P_NP_0}$ are called the *edges* of the polygon. (In general, we need not specify P_0 twice, especially when passing the polygon to the Sutherland-Hodgman clipping algorithm.)

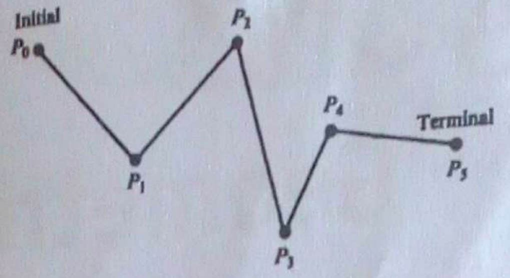


Fig. 9-1

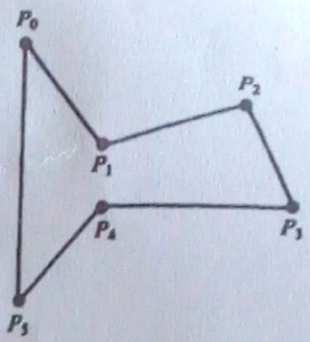
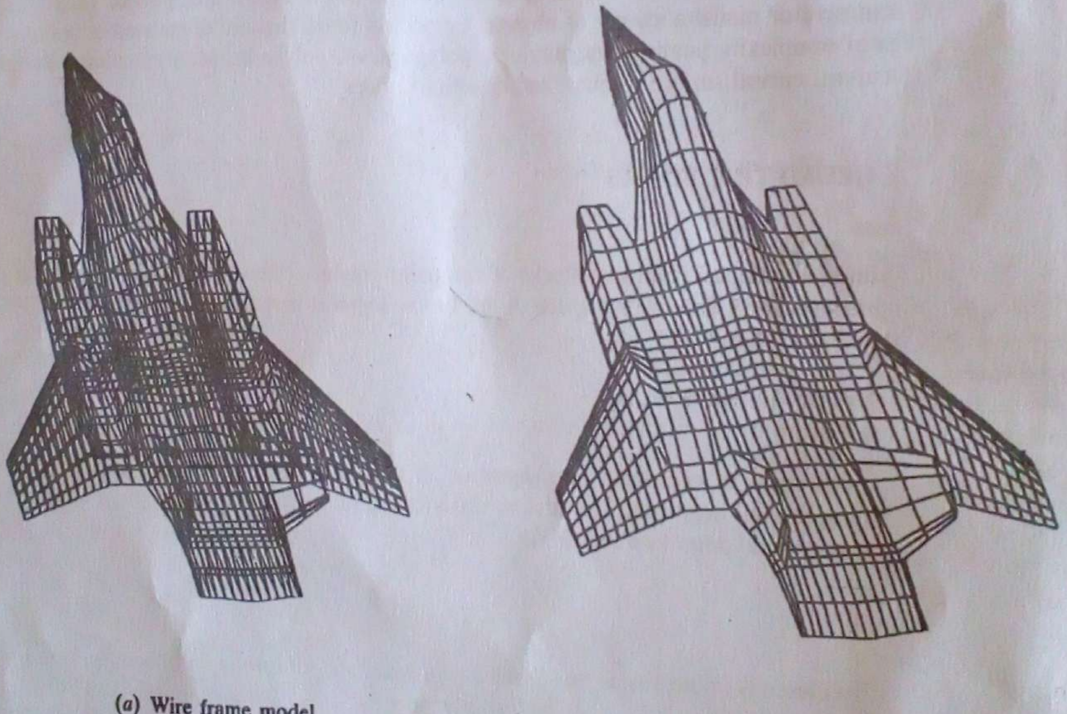


Fig. 9-2

A *planar polygon* is a polygon in which all vertices (and thus the entire polygon) lie on the same plane (see Fig. 9-2).

9.2 WIREFRAME MODELS

A *wirefram model* consists of edges, vertices, and polygons. Here vertices are connected by edges, and polygons are sequences of vertices or edges. The edges may be curved or straight line segments. In the latter case, the wireframe model is called a *polygonal net* or *polygonal mesh* (Fig. 9-3).



(*a*) Wire frame model.

(*b*) Hidden lines removed.

Fig. 9-3

Representing a Polygonal Net Model

There are several different ways of representing a polygonal net model.

1. *Explicit vertex list* $V = \{P_0, P_1, P_2, \ldots, P_N\}$. The points $P_i(x_i, y_i, z_i)$ are the vertices of the polygonal net, stored in the order in which they would be encountered by traveling around the model. Although this form of representation is useful for single polygons, it is quite inefficient for a complete polygonal net, in that shared vertices are repeated several times (see Prob. 9.1). In addition, when displaying the model by drawing the edges, shared edges are drawn several times.
2. *Polygon listing.* In this form of representation, each vertex is stored exactly once in a vertex list $V = (P_0, \ldots, P_N)$, and each polygon is defined by pointing or indexing into this vertex list (see Prob. 9.2). Again, shared edges are drawn several times in displaying the model.
3. *Explicit edge listing.* In this form of representation, we keep a vertex list in which each vertex is stored exactly once and an edge list in which each edge is stored exactly once. Each edge in the edge list points to the two vertices in the vertex list which define that edge. A polygon is now represented as a list of pointers or indices into the edge list. Additional information, such as those polygons sharing a given edge, can also be stored in the edge list (see Prob. 9.9). Explicit edge listing can be used to represent the more general wireframe model. The wireframe model is displayed by drawing all the edges, and each edge is drawn only once.

Polyhedron

A *polyhedron* is a closed polygonal net (i.e., one which encloses a definite volume) in which each polygon is planar. The polygons are called the *faces* of the polyhedron. In modeling, polyhedrons are quite often treated as solid (i.e., block) objects, as opposed to wireframes or two-dimensional surfaces.

Advantages and Disadvantages of Wireframe Models

Wireframe models are used in engineering applications. They are easy to construct and, if they are composed of straight lines, easy to clip and manipulate through the use of geometric and coordinate transformations. However, for building realistic models, especially of highly curved objects, we must use a very large number of polygons to achieve the illusions of roundness and smoothness.

9.3 CURVED SURFACES

The use of curved surfaces allows for a higher level of modeling, especially for the construction of highly realistic models. There are several approaches to modeling curved surfaces. One is an analog of polyhedral models. Instead of using polygons, we model an object by using small, curved *surface patches* placed next to each other. Another approach is to use surfaces that define solid objects, such as polyhedra, spheres, cylinders, and cones. A model can then be constructed with these solid objects used as building blocks. This process is called *solid modeling*.

There are two ways to construct a model—*additive modeling* and *subtractive modeling*. Additive modeling is the process of building the model by assembling many simpler objects. Subtractive modeling is the process of removing pieces from a given object to create a new object, for example, creating a (cylindrical) hole in a sphere or a cube. Subtractive modeling is akin to sculpting.

9.4 CURVE DESIGN

Given $n + 1$ data points, $P_0(x_0, y_0), \ldots, P_n(x_n, y_n)$ we wish to find a curve that, in some sense, fits the shape outlined by these points. If we require the curve to pass through all the points, we are faced with the problem of *interpolation*. If we require only that the curve be near these points, we are faced with the problem of *approximation*. Interpolation arises, for example, in reconstructing the shape of a digitiz curved objects. Approximation is used in computer graphics to design curves that "look good" or must meet some aesthetic goal. To solve these often quite distinct problems, it is necessary to find ways of building curves out of smaller pieces, or *curve segments*, in order to meet the design criteria. (Note that curves and curve segments can be modeled as polylines, i.e., drawn with extremely short line segments.) When modeling a curve $f(x)$ by using curve segments, we try to represent the curve as a sum of smaller segments $\Phi_i(x)$ (called *basis* or *blending functions*):

$$f(x) = \sum_{i=0}^{N} a_i \Phi_i(x)$$

We choose these blending functions with an eye toward computation and display. For this reason, polynomials are often the blending functions of choice.

A *polynomial of degree n* is a function that has the form

$$Q(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

This polynomial is determined by its $n+1$ *coefficients* $[a_n, \ldots, a_0]$.

A *continuous piecewise polynomial* $Q(x)$ of degree n is a set of k polynomials $q_i(x)$, each of degree n and $k+1$ *knots* (nodes) $t_0, \ldots, t_k$ so that

$$Q(x) = q_i(x) \qquad \text{for} \qquad t_i \le x \le t_{i+1} \qquad \text{and} \qquad i = 0, \ldots, k-1$$

Note that this definition requires the polynomials to match or piece together at the knots, that is, $q_{i-1}(t_i) = q_i(t_i)$, $i = 1, \ldots, k-1$. This requirement imposes no restrictions on how smoothly the polynomials $q_i(x)$ fit together. For example, there can be corners or sharp contours at the knots (see Fig. 9-4).

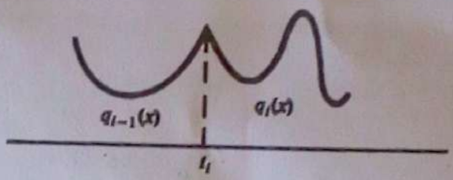


Fig. 9-4

Polynomials of high degree are not very useful for curve designing because of their oscillatory nature.

The most useful piecewise polynomials are those for which the polynomials $q_i(x)$ are cubic (degree 3). There are several reasons for this. One is that the piecewise cubic closely resembles the way a drafter uses a mechanical spline. In addition, 3 is the smallest degree which has the required smoothness properties for describing pleasing shapes. It is also the minimum number needed to represent three-dimensional curves.

9.5 POLYNOMIAL BASIS FUNCTIONS

Let $P_0(x_0, y_0), \ldots, P_n(x_n, y_n)$ represent $n+1$ data points. In addition, let $t_0, t_1, t_2, \ldots,$ be any numbers (called *knots*). The following are common choices for basis or blending functions.

Lagrange Polynomials of Degree *n*

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

Note that $L_i(x_i) = 1$ and $L_i(x_j) = 0$ for all $j \neq i$. (Here Π represents term-by-term multiplication.)

Hermite Cubic Polynomials

Refer to Fig. 9-5.

$$H_i(x) = \begin{cases} -\dfrac{2(x - t_{i-1})^3}{(t_i - t_{i-1})^3} + \dfrac{3(x - t_{i-1})^2}{(t_i - t_{i-1})^2} & t_{i-1} \le x \le t_i \\[2ex] -\dfrac{2(t_{i+1} - x)^3}{(t_{i+1} - t_i)^3} + \dfrac{3(t_{i+1} - x)^2}{(t_{i+1} - t_i)^2} & t_i \le x \le t_{i+1} \end{cases}$$

$$\bar{H}_i(x) = \begin{cases} \dfrac{(x - t_{i-1})^2(x - t_i)}{(t_i - t_{i-1})^2} & t_{i-1} \le x \le t_i \\[2ex] \dfrac{(x - t_i)(t_{i+1} - x)^2}{(t_{i+1} - t_i)^2} & t_i \le x \le t_{i+1} \end{cases}$$

B-Splines

Refer to Fig. 9-6. For the knot set $t_0, t_1, t_2, \ldots$, the nth-degree B-splines $B_{i,n}$ are defined recursively:

$$B_{i,0}(x) = \begin{cases} 1 & t_i \le x \le t_{i+1} \\ 0 & \text{otherwise} \end{cases} \qquad B_{i,n}(x) = \frac{x - t_i}{t_{i+n} - t_i} B_{i,n-1}(x) + \frac{t_{i+n+1} - x}{t_{i+n+1} - t_{i+1}} B_{i+1,n-1}(x)$$

for $t_i \le x \le t_{i+n+1}$. Note that $B_{i,n}(x)$ is nonzero only in the interval $[t_i, t_{i+n+1}]$. In particular, the cubic B-spline $B_{i,3}$ is nonzero over the interval $[t_i, t_{i+4}]$ (which spans the knots $t_i, t_{i+1}, t_{i+2}, t_{i+3}, t_{i+4}$). In addition, for nonrepeated knots, the B-spline is zero at the endknots t_i and t_{i+n+1} (from Prob. 9.3), that is,

$$\begin{aligned} B_{i,n}(t_i) &= 0 \\ B_{i,n}(t_{i+n+1}) &= 0 \end{aligned} \qquad (n \ge 1)$$

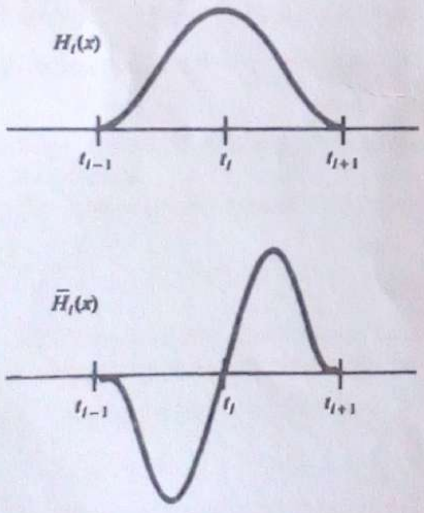


Fig. 9-5 Hermite cubic basis functions.

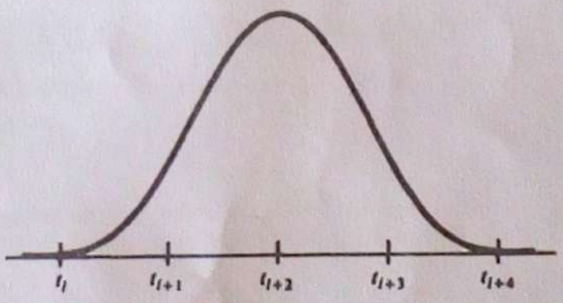


Fig. 9-6 Cubic B-spline $B_{i,3}(x)$.

In using B-splines, we allow for repeated knots, that is, $t_i = t_{i+1} = \cdots$. This means that $B_{i,n}$ can have the form $\frac{0}{0}$ in its definition. By letting $\frac{0}{0} = 0$, we extend the definition of $B_{i,n}$ to incorporate repeated knots.

Bernstein Polynomials

Refer to Fig. 9-7. The Bernstein polynomials of degree n over the interval [0, 1] are defined as

$$BE_{k,n}(x) = \frac{n!}{k!(n-k)!} x^k (1-x)^{n-k}, \qquad 0 \le x \le 1$$

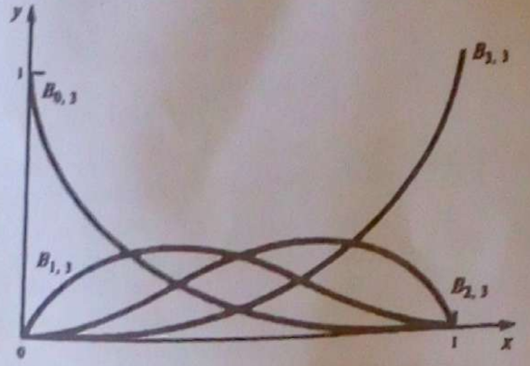


Fig. 9-7 Cubic Bernstein polynomials.

The cubic Bernstein polynomials are

$$\begin{aligned} B_{0,3}(x) &= 1 - 3x + 3x^2 - x^3 \\ B_{1,3}(x) &= 3(x - 2x^2 + x^3) \\ B_{2,3}(x) &= 3(x^2 - x^3) \\ B_{3,3}(x) &= x^3 \end{aligned}$$

9.6 THE PROBLEM OF INTERPOLATION

Given data points $P_0(x_0, y_0), \ldots, P_n(x_n, y_n)$, we wish to find a curve which passes through these points.

Lagrange Polynomial Interpolation Solution

Here

$$L(x) = \sum_{i=0}^{n} y_i L_i(x)$$

where $L_i(x)$ are the Lagrange polynomials and $L(x)$ is the nth-degree polynomial interpolating the data points.

Hermitian Cubic Interpolation Solution

We wish to find a piecewise polynomial $H(x)$ of degree 3 which passes through the data points and is also continuously differentiable at these points. We also prescribe the values of the derivatives y' (or slope of the tangent line) at the given data points, that is, we prescribe the points $(x_0, y'_0), \ldots, (x_n, y'_n)$:

$$H(x) = \sum_{i=0}^{n} [y_i H_i(x) + y'_i \bar{H}_i(x)]$$

where $H_i(x)$ and $\bar{H}_i(x)$ are the Hermitian cubic basis functions and $t_0 = x_0, t_1 = x_1, t_2 = x_2, \ldots, t_n = x_n$ are the choices for the knot set.

Spline Interpolation

If we require that the interpolating piecewise polynomial be joined as smoothly as possible at the data points, the resulting curve is called a *spline*. Therefore, a spline of degree m has continuous derivatives up to order $m - 1$ at the data points.

It can be shown that any mth-degree spline that passes through $n + 1$ data points can be represented in terms of the B-spline basis functions $B_{i,n}$ as

$$S_m(x) = \sum_{i=0}^{m+n-1} a_i B_{i,m}(x)$$

In order to define the B-spline functions $B_{i,m}(x)$ so as to solve the interpolation problem, the knots $t_0, t_1, \ldots, t_{m+n+1}$ must be chosen to satisfy the Shoenberg–Whitney condition:

$$t_i < x_i < t_{i+m+1}, \qquad i = 0, \ldots, n$$

The following choices for the knots satisfy this condition (see Prob. 9.4):

Step 1. Choose

$$t_0 = \cdots = t_m < x_0 \qquad t_{n+1} = \cdots = t_{m+n+1} > x_n$$

Step 2. Choose the remaining knots according to

$$t_{i+m+1} = \frac{x_{i+1} + \cdots + x_{i+m}}{m}, \qquad i = 0, \ldots, n - m - 1$$

For cubic splines ($m = 3$), an alternative to step 2, requiring less computation, is step 2′. Choose

$$t_{i+4} = x_{i+2}, \qquad i = 0, \ldots, n - 4$$

The splines $S_2(x)$ and $S_3(x)$ are called *quadratic* and *cubic splines*, respectively:

$$S_2(x) = \sum_{i=0}^{n+1} a_i B_{i,2}(x) \qquad \text{and} \qquad S_3(x) = \sum_{i=0}^{n+2} a_i B_{i,3}(x)$$

Confining our attention to the cubic spline, we see that there are $n + 3$ coefficients a_i to evaluate, requiring $n + 3$ equations.

The interpolation criterion $S_3(x_j) = y_j, j = 0, \ldots, n$ provides $n + 1$ equations:

$$y_j = S_3(x_j) = \sum_{i=0}^{n+2} a_i B_{i,3}(x_j)$$

The remaining two equations are usually specified as boundary conditions at the endpoints x_0 and x_n. Some choices for boundary conditions are

1. *Natural spline condition*

$$S''_3(x_0) = 0 \qquad S''_3(x_n) = 0$$

2. *Clamped spline condition*

$$S'_3(x_0) = y'_0 \qquad S'_3(x_n) = y'_n$$

where y'_0 and y'_n are prescribed derivative values.

3. *Cyclic spline condition*

$$S_3'(x_0) = S_3'(x_n) \qquad S_3''(x_0) = S_3''(x_n)$$

This is useful for producing closed curves.

4. *Anticyclic spline condition*

$$S_3'(x_0) = -S_3'(x_n) \qquad S_3''(x_0) = -S_3''(x_n)$$

This is useful in producing splines with parallel endings whose tangent vectors are equal in magnitude but opposite in direction.

For technical reasons, boundary condition 1, the so-called natural boundary condition, is the least preferred choice.

9.7 THE PROBLEM OF APPROXIMATION

The problem is to provide a smooth representation of a three-dimensional curve which approximates given data so as to yield a given shape. Usually the data is given interactively in the form of a guiding polyline determined by *control points* $P_0(x_0, y_0, z_0), P_1(x_1, y_1, z_1), \ldots, P_n(x_n, y_n, z_n)$. We would like to find a curve which approximates the shape of this guiding polyline (see Fig. 9-8).

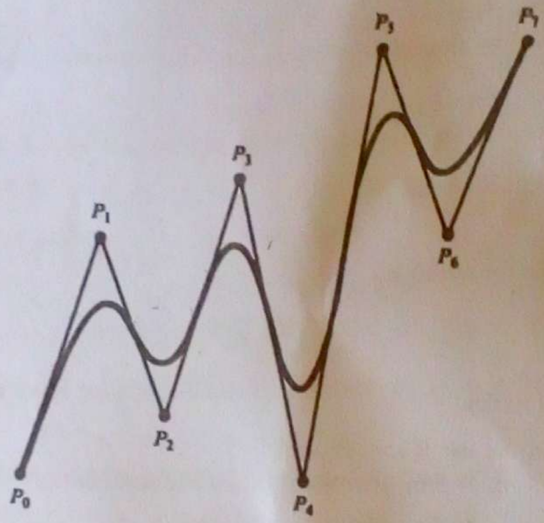


Fig. 9-8

Bézier–Bernstein Approximation

Using the Bernstein polynomials, we form the parametric curves:

$$P(t): \begin{cases} x(t) = \sum_{i=0}^{n} x_i BE_{i,n}(t) \\ y(t) = \sum_{i=0}^{n} y_i BE_{i,n}(t) \qquad 0 \le t \le 1 \\ z(t) = \sum_{i=0}^{n} z_i BE_{i,n}(t) \end{cases}$$

where $P(t)$ is called the *Bézier curve.*

Properties of the Bézier–Bernstein Approximation

There are four basic properties:

1. The Bézier curve has the same endpoints as the guiding polyline, that is:

$$P_0 = P(0) = [x(0), y(0), z(0)] \qquad P_n = P(1) = [x(1), y(1), z(1)]$$

2. The direction of the tangent vector at the endpoints P_0, P_n is the same as that of the vector determined by the first and last segments $\overline{P_0P_1}$, $\overline{P_{n-1}P_n}$ of the guiding polyline. In particular $P'(0) = n \cdot (P_1 - P_0)$ [i.e., $x'(0) = n(x_1 - x_0)$, $y'(0) = n(y_1 - y_0)$, $z'(0) = n(z_1 - z_0)$] and $P'(1) = n \cdot (P_n - P_{n-1})$.
3. The Bézier curve lies entirely within the convex hull of the guiding polyline. In two dimensions, the *convex hull* is the polygon formed by placing a "rubber band" about the collection of points $P_0, \ldots, P_n$.
4. Bézier curves are suited to interactive design. In fact, Bézier curves can be pieced together so as to ensure continuous differentiability at their juncture by letting the edges of the two different guiding polylines that are adjacent to the common endpoint be collinear (see Fig. 9-9).

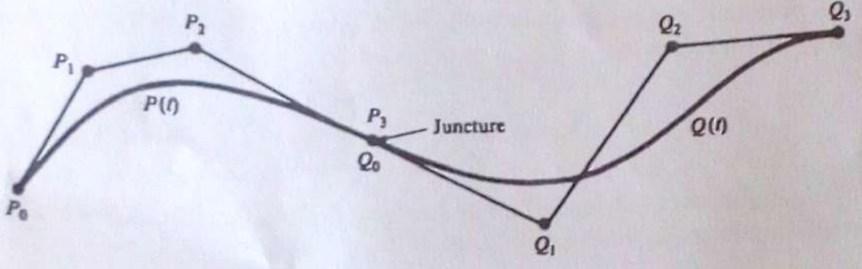


Fig. 9-9

Bézier–B-Spline Approximation

For this approximation, we use B-splines (see Fig. 9-10)

$$P(t): \begin{cases} x(t) = \sum_{i=0}^{n} x_i B_{i,m}(t) \\ y(t) = \sum_{i=0}^{n} y_i B_{i,m}(t) \qquad 0 \le t \le n - m + 1 \\ z(t) = \sum_{i=0}^{n} z_i B_{i,m}(t) \end{cases}$$

The mth-degree B-splines $B_{i,m}(t)$, $i = 0, \ldots, n$, are defined for t in the parameter range $[0, n - m + 1]$. The knot set $t_0, \ldots, t_{n+m+1}$ is chosen to be the set $\underbrace{0, \ldots, 0}_{m+1}, 1, 2, \ldots, n - m, \underbrace{n - m + 1, \ldots, n - m + 1}_{m+1}$. This use of repeated knots ensures that the endpoints of the spline coincide with the endpoints of the guiding polyline (Prob. 9.6).

Since the knot spacing is uniform, we can also use an explicit form for calculating the B-splines (Prob. 9.10).

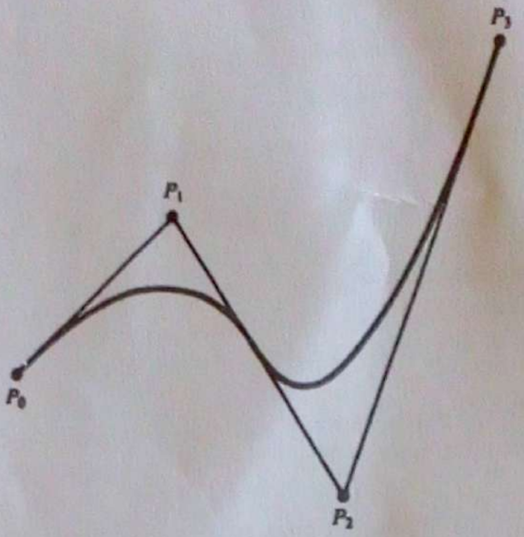


Fig. 9-10 Cubic Bézier B-spline.

Closed Curves

To construct a closed B-spline curve which approximates a given closed guiding polygon, we need only choose the knots $t_0, \ldots, t_{n+m+1}$ to be cyclic, i.e., $[0, 1, \ldots, n, 0, 1, \ldots]$. So

$$t_{m+1} = t_0 = 0 \qquad t_{m+2} = t_1 = 1 \qquad t_{m+1+i} = t_i$$

In practice, the quadratic and cubic B-splines $B_{i,2}$ and $B_{i,3}$ are the easiest to work with and provide enough flexibility for use in a wide range of curve design problems.

Properties of Bézier–B-Spline Approximation

There are five basic properties:

1. The Bézier–B-spline approximation has the same properties as the Bézier–Bernstein approximation; in fact, they are the same piecewise polynomial if $m = n$. This includes the properties of agreement with the guiding polygon and the tangent vectors at the endpoints and the convex hull property.
2. If the guiding polyline has $m + 1$ consecutive vertices (control points) which are collinear, the resulting span of the Bézier–B-spline will be linear. So this approximation allows for linear sections to be embedded within the curve.
3. The Bézier–B-spline approximation provides for the local control of curve shape. If a single control point is changed, portions of the curve that lie far away are not disturbed. In fact, only $m + 1$ of the spans are affected. This is due to the local nature of the B-spline basis functions.
4. Bézier–B-splines produce a closer fit to the guiding polygon than does the Bézier–Bernstein approximation.
5. The Bézier–B-spline approximation allows the use of control points P_i counted with *multiplicities* of 2 or more. That is, $P_i = P_{i+1} = \cdots = P_{i+k}$ for $k \geq 1$. This results in an approximation which is pulled closer toward this control point. In fact, if the point has multiplicity $m + 1$, the curve will pass through it.

9.8 CURVED-SURFACE DESIGN

The modeling and approximation of curved surfaces is difficult and involves many complex issues. We will look at two methods for representing a surface: *guiding nets* and *interpolating surface patches*.

Guiding Nets

This technique is a direct generalization of the Bézier–Bernstein and Bézier–B-spline approximation methods for curves. A *guiding net* is a polygonal net with vertices $P_{ij}(x_{ij}, y_{ij}, z_{ij})$, $i = 0, \ldots, m$, and $j = 0, \ldots, n$ (see Fig. 9-11).

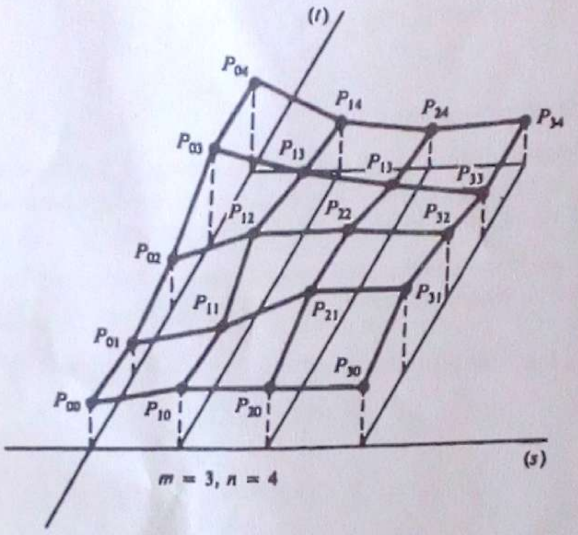


Fig. 9-11

1. *Bézier–Bernstein surface.* This is the surface with parametric equations:

$$Q(s,t): \begin{cases} x(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} x_{ij} BE_{i,m}(s) BE_{j,n}(t) \\ y(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} y_{ij} BE_{i,m}(s) BE_{j,n}(t) \\ z(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} z_{ij} BE_{i,m}(s) BE_{j,n}(t) \end{cases}$$

Here $0 \le s,\ t \le 1$ and BE are the Bernstein polynomials. This approximation has properties analogous to the one-dimensional case with respect to the corner points P_{00}, P_{m0}, P_{0n}, and P_{mn}. The convex hull property is also satisfied.

2. *Bézier–B-spline approximation.* In parametric form this is expressed as

$$Q(s,t): \begin{cases} x(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} x_{ij}B_{i,\alpha}(s)B_{j,\beta}(t) \\ y(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} y_{ij}B_{i,\alpha}(s)B_{j,\beta}(t) \\ z(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} z_{ij}B_{i,\alpha}(s)B_{j,\beta}(t) \end{cases}$$

where $0 \le s \le m - \alpha + 1$ and $0 \le s \le n - \beta + 1$. The knot sets for s and t used to define the B-splines $B_{i,\alpha}(s)$ and $B_{j,\beta}(t)$ are determined as in the one-dimensional case.

Quadratic approximation occurs when $\alpha = \beta = 2$. Cubic approximation occurs when $\alpha = \beta = 3$. In general, quadratic or cubic B-splines are most often used. For both these methods, the construction of the guiding net (by locating the control points P_{ij}) is left to the user.

Interpolating Surface Patches

Instead of using a given set of points P_{ij} to construct a given surface, the process of interpolating surface patches is based upon prescribing boundary curves for a surface patch and "filling in" the interior of the patch by interpolating between the boundary curves.

1. *Coons surfaces.* For this technique, a patch is determined by specifying four bounding curves, denoted in parametric vector form as $P(s, 0)$, $P(s, 1)$, $P(0, t)$, and $P(1, t)$, $0 \le s, t \le 1$ (see Fig. 9-12).

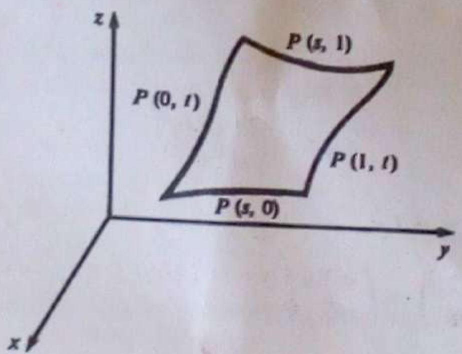


Fig. 9-12

The (linear) *Coons surface patch* interpolating the boundary curves can be written in vector form by using linear interpolation (or blending):

$$Q(s,t) = P(s,0)(1-t) + P(s,1)t + P(0,t)(1-s) + P(1,t)s - P(0,0)(1-s)(1-t) - P(0,1)(1-s)t - P(1,0)s(1-t) - P(1,1)st$$

(The subtractions are required so that the interpolators between corner points are not counted twice.) This idea can be extended to define more general surface patches.

2. *Lofted surfaces. Lofting* is used where the surface to be constructed stretches in a given direction; an example is the hull of a ship.

Given two or more space curves, called *cross-section curves*, *lofting* is the process of blending the cross sections together using longitudinal blending curves. The simplest example is *linear blending* between cross-section curves $P_1(s)$ and $P_2(s)$. The lofted surface is $Q(s, t) = (1 - t)P_1(s) + tP_2(s)$ (see Fig. 9-13).

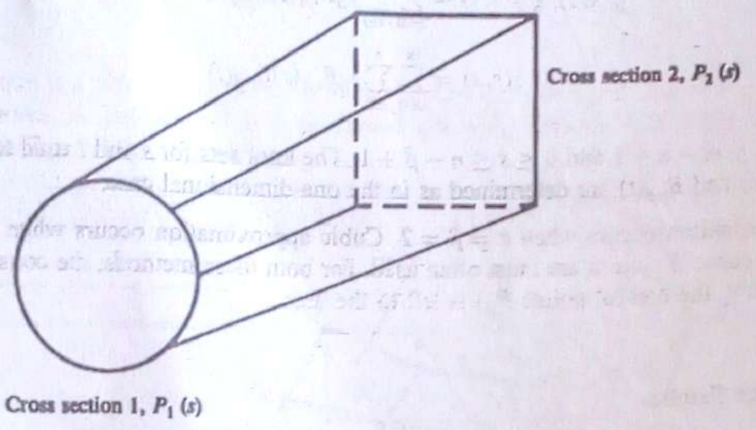


Fig. 9-13

9.9 TRANSFORMING CURVES AND SURFACES

All the curve and surface models that we have constructed have the general form

$$x = \Sigma x_i \Phi_i \quad \text{and} \quad y = \Sigma y_i \Phi_i$$

(Add a z component for three dimensions.)

If M is 2×2 transformation matrix

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

we can apply M to the functions x and y:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \Sigma x_i \Phi_i \\ \Sigma y_i \Phi_i \end{pmatrix} = \begin{pmatrix} a(\Sigma x_i \Phi_i) + b(\Sigma y_i \Phi_i) \\ c(\Sigma x_i \Phi_i) + d(\Sigma y_i \Phi_i) \end{pmatrix} = \begin{pmatrix} \Sigma(ax_i + by_i)\Phi_i \\ \Sigma(cx_i + dy_i)\Phi_i \end{pmatrix}$$

The transformed functions are then

$$\bar{x} = \Sigma(ax_i + by_i)\Phi_i \qquad \bar{y} = \Sigma(cx_i + dy_i)\Phi_i$$

In other words, to transform these curves and surfaces, it is necessary only to transform the coefficients (x_i, y_i). In most cases these coefficients represent data or control points. So the transformation of the approximation of a curve or surface is found by first transforming the control points and then forming the approximation based on the transformed points.

9.10 QUADRIC SURFACES

Spheres, cylinders, and cones are part of the family of surfaces called *quadric surfaces*. A quadric surface is defined by an equation which is of the second degree (in x, y, or z).

The canonical quadric surfaces are as follows:

Sphere

From Fig. 9-14:

$$x^2 + y^2 + z^2 = R^2$$

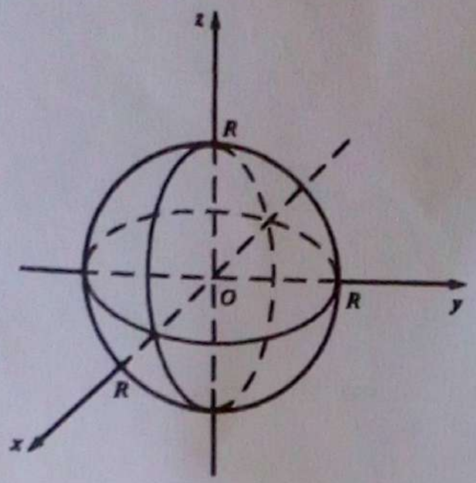


Fig. 9-14

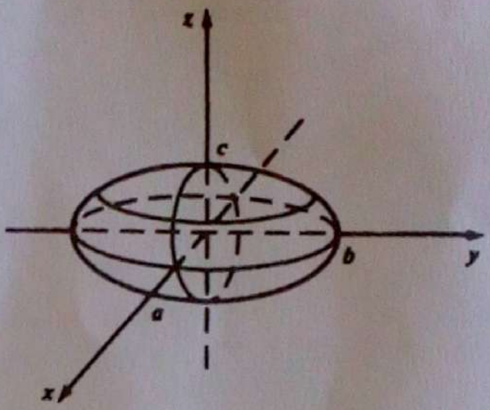


Fig. 9-15

Ellipsoid

From Fig. 9-15:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

One-sheeted Hyperboloid

From Fig. 9-16:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$$

Two-sheeted Hyperboloid

From Fig. 9-17:

$$\frac{z^2}{c^2} - \frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

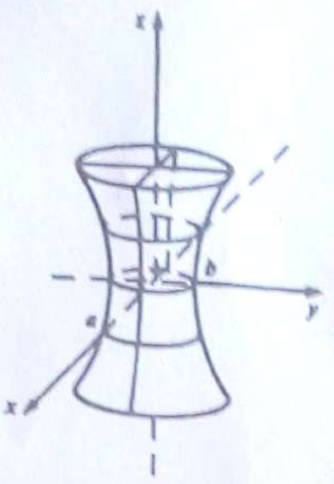


Fig. 9-16

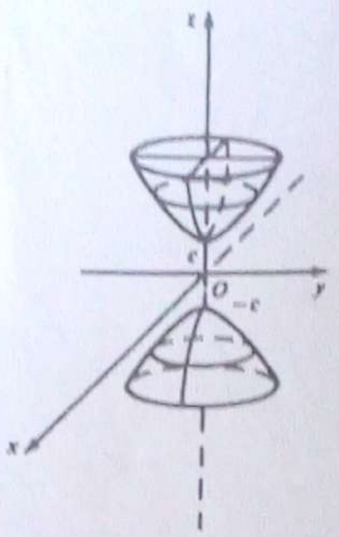


Fig. 9-17

Elliptic Cylinder

From Fig. 9-18:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$

When $a=b=R$, the cylinder is a circular cylinder of radius R.

Elliptic Paraboloid

From Fig. 9-19:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=cz$$

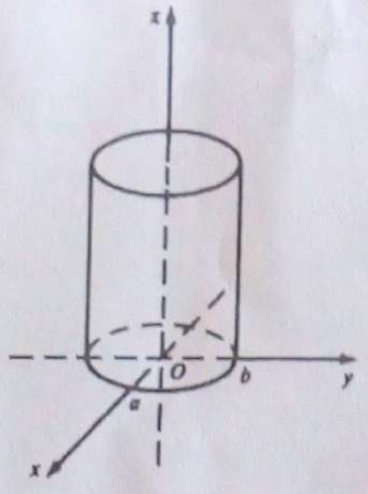


Fig. 9-18

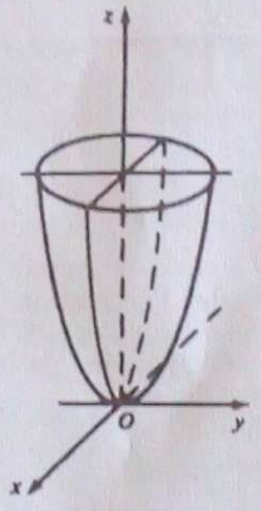


Fig. 9-19

Hyperbolic Parboloid

From Fig. 9-20:

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = cz$$

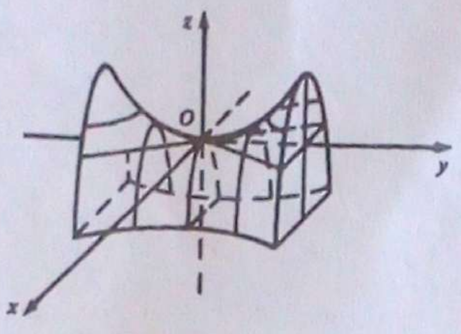


Fig. 9-20

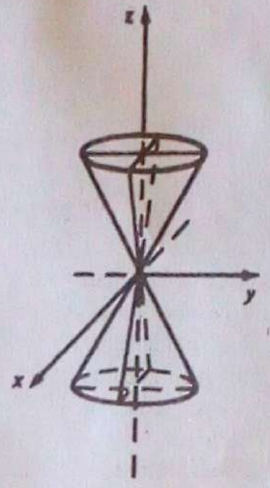


Fig. 9-21

Elliptic Cone

From Fig. 9-21:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = z^2$$

When $a = b = R$, we have a right circular cone. If we restrict z so that $z > 0$, we have the upper cone.

Note that, except for spheres and ellipsoids, the mathematical definition produces a figure of infinite extent. To use it for computer graphics, we must specify bounds for the surface in the form of bounding clipping planes having the form $z = h$.

~~9.11 EXAMPLE: TERRAIN GENERATION~~

We show two different ways to generate a *triangle mesh* (i.e., a polynomial net consisting of triangles) that models the random peaks and valleys of a mountainous landscape.

Midpoint Displacement

This is a recursive approach that begins with one or more triangles over the terrain area [see Fig. 9-22(*a*)]. We use the midpoints of the edges to subdivide each triangle into four smaller ones. The midpoints are randomly elevated to produce a rugged appearance [see Fig. 9-22(*b*)]. The coordinates (x_m, y_m, z_m) of the midpoint of the edge between $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$ are calculated as follows:

$$\begin{cases} x_m = (x_2 - x_1)/2 \\ y_m = (y_2 - y_1)/2 + r \\ z_m = (z_2 - z_1)/2 \end{cases}$$