CHAPTER 9

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.)

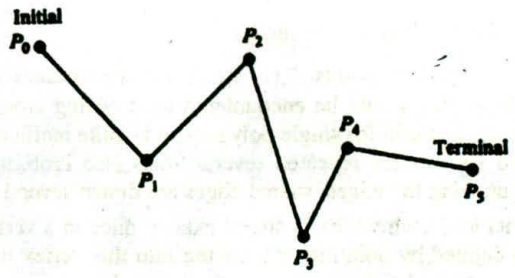


Fig. 9-1

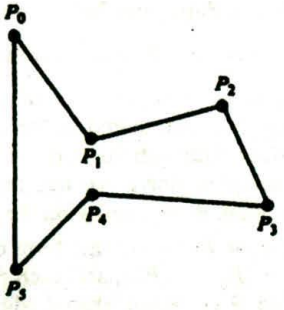
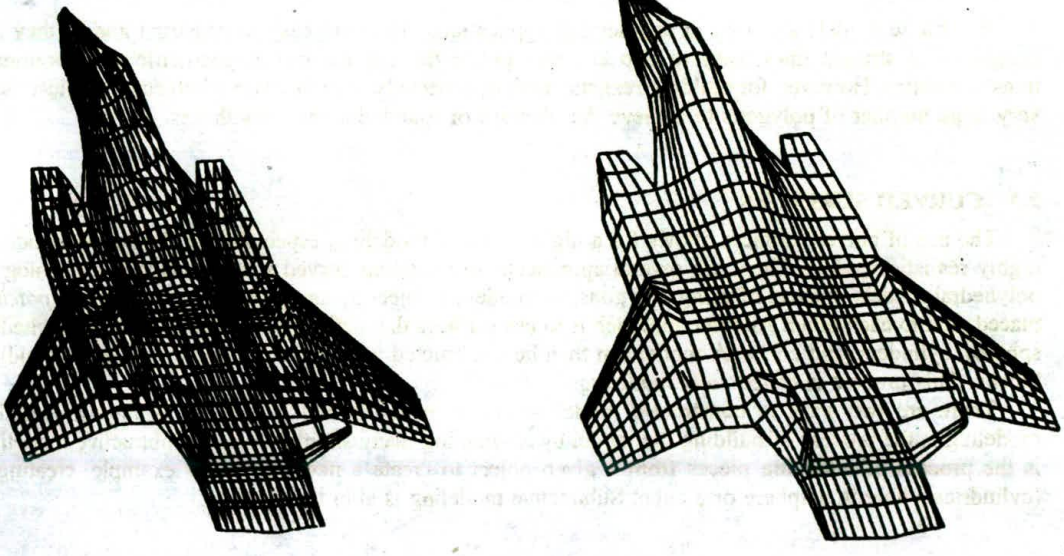


Fig. 9-2

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 constructiodn 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 digitized 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).

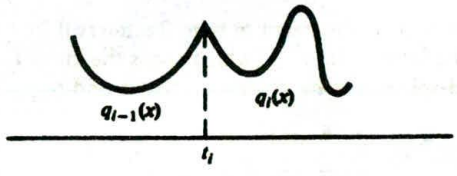


Fig. 9-4

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.

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

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

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,

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

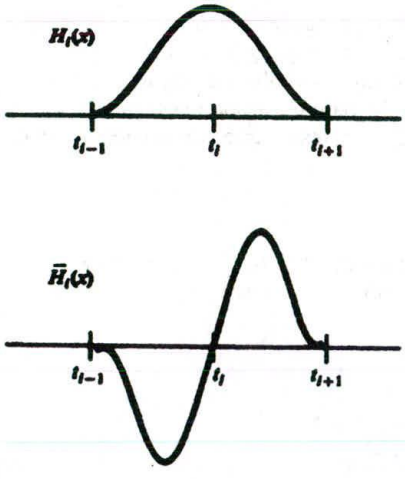


Fig. 9-5 Hermite cubic basis functions.

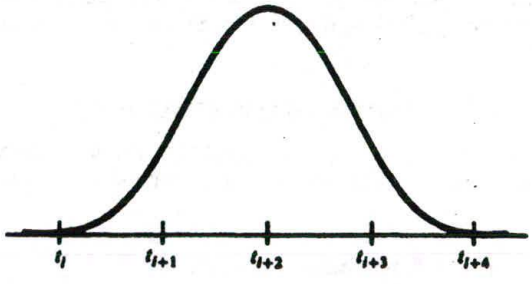


Fig. 9-6 Cubic B-spline $B_{i,3}(x)$.

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

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

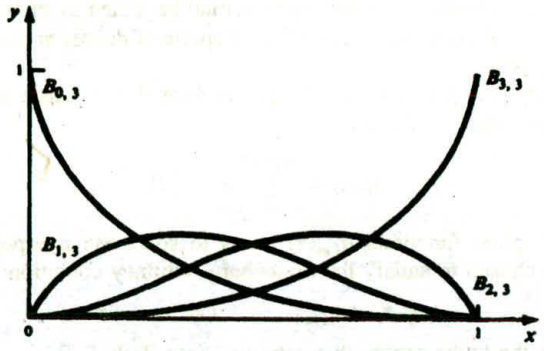


Fig. 9-7 Cubic Bernstein polynomials.

The cubic Bernstein polynomials are

$$B_{0,3}(x) = 1 - 3x + 3x^2 - x^3$$
$$B_{1,3}(x) = 3(x - 2x^2 + x^3)$$
$$B_{2,3}(x) = 3(x^2 - x^3)$$
$$B_{3,3}(x) = x^3$$

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).

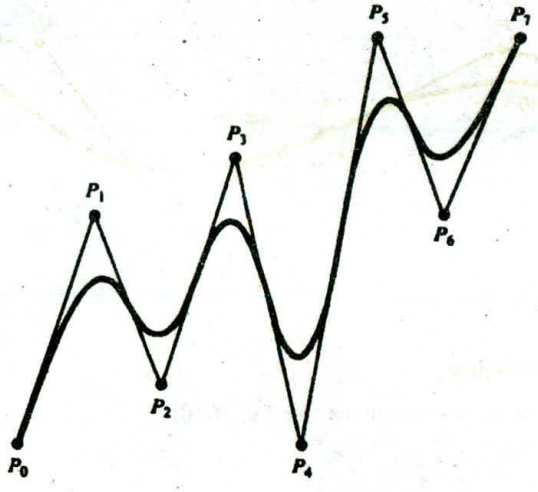


Fig. 9-8

Bézier–Bernstein Approximation

Using the Bernstein polynomials, we form the parametric curves:

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

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).

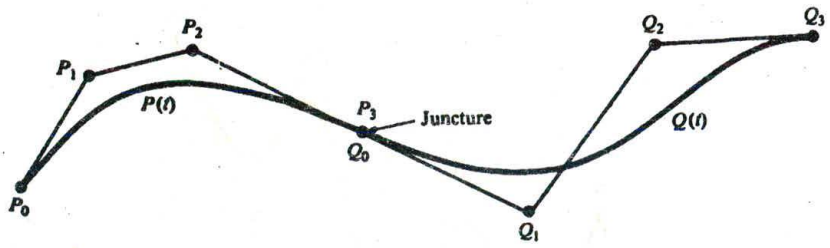


Fig. 9-9

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).

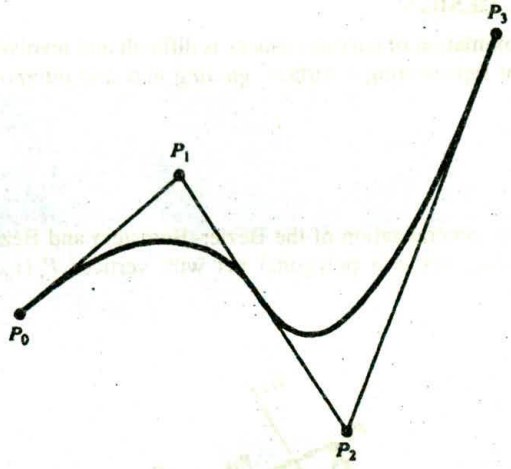


Fig. 9-10 Cubic Bézier B-spline.

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).

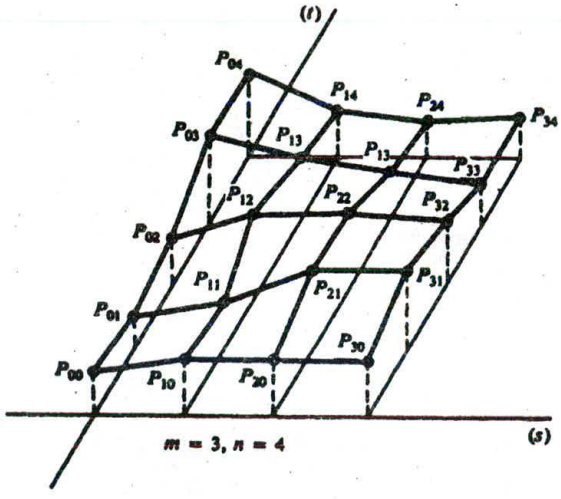


Fig. 9-11

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).

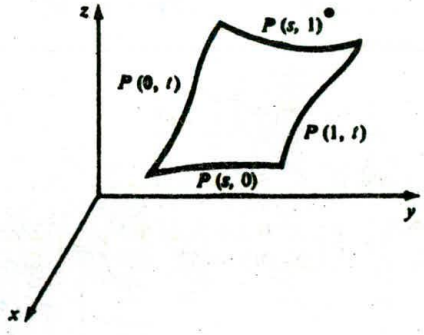


Fig. 9-12

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

$$\begin{aligned} Q(s,t) = {} & P(s,0)(1-t)+P(s,1)t+P(0,t)(1-s)+P(1,t)s-P(0,0)(1-s)(1-t) \\ & -P(0,1)(1-s)t-P(1,0)s(1-t)-P(1,1)st \end{aligned}$$

(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).

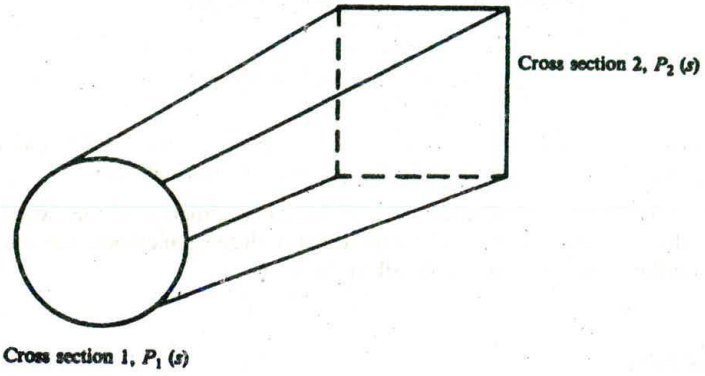


Fig. 9-13

9.9 TRANSFORMING CURVES AND SURFACES

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

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

(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

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

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:

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

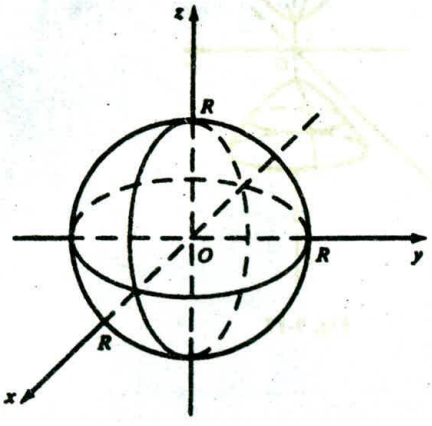


Fig. 9-14

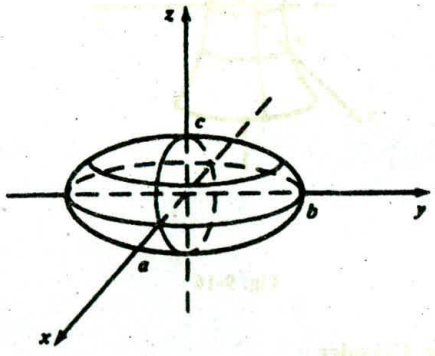


Fig. 9-15

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:

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

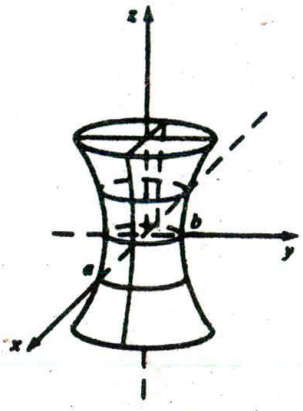


Fig. 9-16

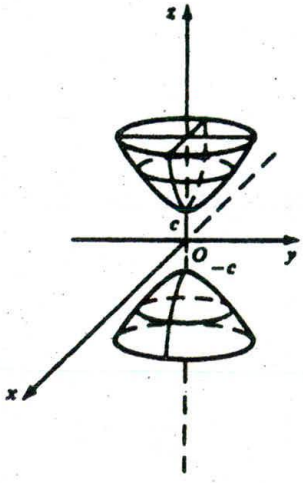


Fig. 9-17

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:

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

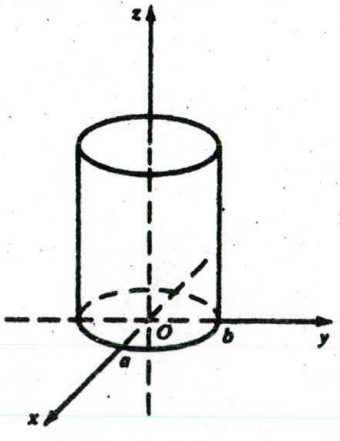


Fig. 9-18

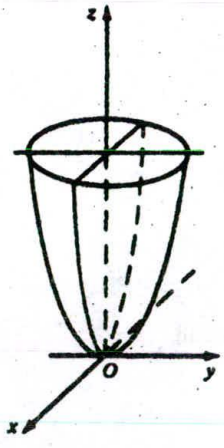


Fig. 9-19

Hyperbolic Parboloid

From Fig. 9-20:

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

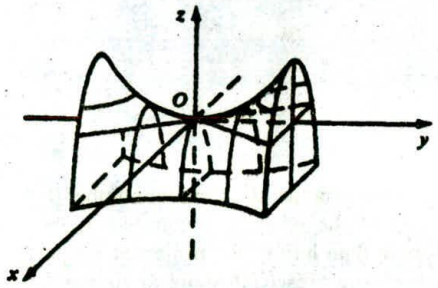


Fig. 9-20

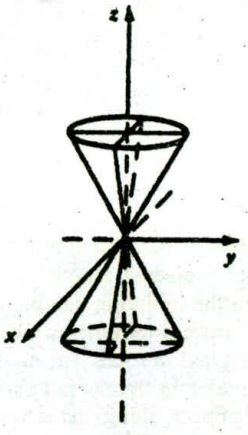


Fig. 9-21

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:

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

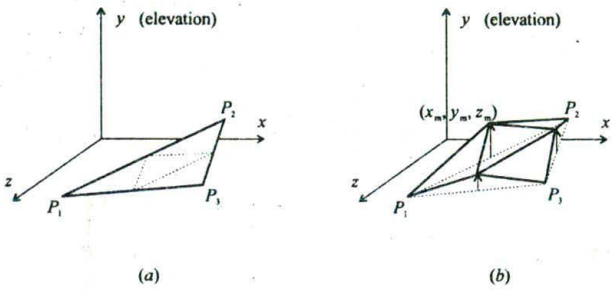


Fig. 9-22

where r is a random displacement that can be expressed as the product of a random number $0 \leq \lambda \leq 1$ and a function of the projected length (onto the x–z plane) of the edge. For example, if the function is $F(\rho) = \rho/2$, then the midpoint can be elevated by no more than half of the projected length of the edge. The subdivision process terminates when all edges project to a preset threshold or shorter length.

Since the original vertices are not moved and each midpoint is displaced only once, the resulting terrain tends to maintain the overall shape of the initial model. On the other hand, deep valleys that look unrealistic may appear along the original edges. We can alleviate this problem by adding a random displacement to the vertices as well as the midpoints at each recursion stage. The tradeoff here is less control over the shape of the final terrain structure.

Brownian Distribution of Disks

In this approach we superimpose a grid onto the terrain area and initialize a counter for each grid cell (see Fig. 9-23 for a rectangular area that can be represented by a two-dimensional counter array). We then randomly move a circular disk over the area. At each disk position we increment the counter for every grid cell that is covered by the disk. After a while the counter values form an elevation profile of a pile of randomly placed disks. Each elevation value describes the height of a point/vertex above the corresponding grid cell. A triangle-mesh model of the resulting terrain can thus be constructed by first connecting each vertex to its four neighbors [analogous to the four neighbouring pixels in Fig. 3-18(*a*)] to form a quadrilateral-mesh and then adding diagonal edges to divide each quadrilateral into two triangles.

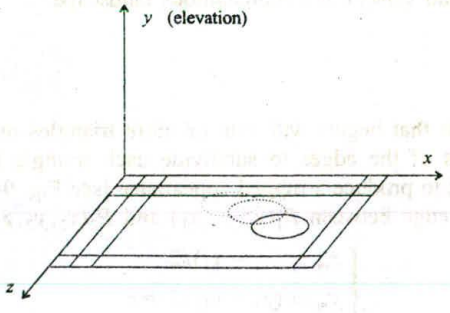


Fig. 9-23

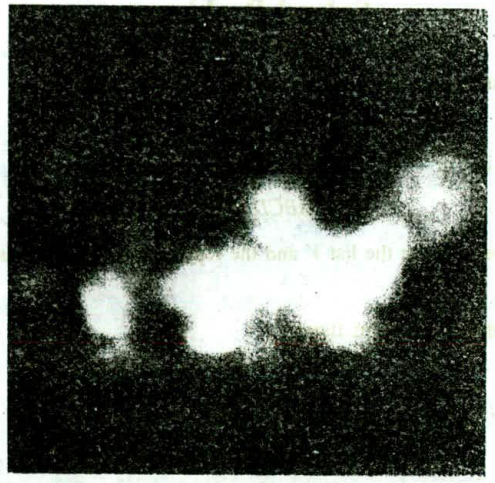


Fig. 9-24

For example, the gray-scale image in Fig. 9-24 represents the values in a 512×512 counter array. The values are obtained using a disk of radius 25 after 6000 random movements, each of which is a step of size 3 forward or backward and simultaneously a step of the same size to the left or to the right. Whenever the center of the disk moves beyond the image boundary it is reset to the middle of the image. The counter values are normalized so that the maximum count corresponds to white in the image. The resultant triangle-mesh is viewed in Fig. 9-25 using perspective projection, looking from a point near the lower-right corner of Fig. 9-24 towards the center (with hidden surfaces removed and visible surfaces lit by a light source).

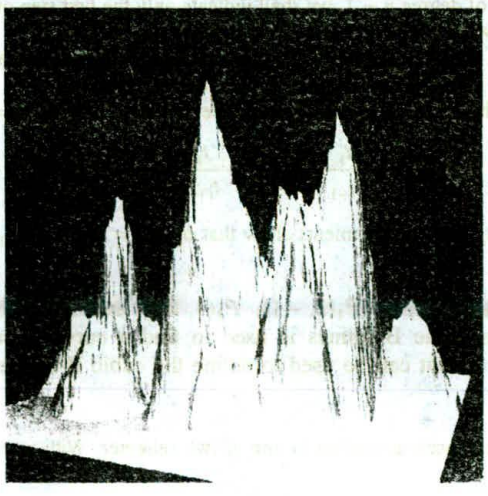


Fig. 9-25

Solved Problems

9.1 Represent a cube by using an explicit vertex list.

SOLUTION

Using the notation in Fig. 7-12, one possible representation is

$$V = \{ABCDAFEDCHEFGHGB\}$$

Note the vertex repetitions in the list V and the repetition of edge FE as EF.

9.2 Represent a cube by using polygon listing.

SOLUTION

Referring to Fig. 7-12, we form the vertex list

$$V = \{ABCDEFGH\}$$

The faces of the cube are polygons (which, in this case, are squares) $P_1, \ldots, P_6$, where

$$P_1 = \{ABCD\} \qquad P_4 = \{ABGF\}$$
$$P_2 = \{CDEH\} \qquad P_5 = \{BCHG\}$$
$$P_3 = \{ADEF\} \qquad P_6 = \{EFGH\}$$

Note the edge repetitions encountered in drawing the polygons. For example, polygons P_2 and P_3 share the edge DE.

9.3 Show that the nth-degree B-spline basis functions $B_{i,n}(x)$ satisfy

$$B_{i,n}(x) = 0 \quad \text{if} \quad x < t_i \quad \text{or} \quad x > t_{i+n+1}$$

SOLUTION

Since the B-spline basis functions $B_{i,n}(x)$ are defined recursively in terms of the lower-order B-splines $B_{i,n-1}(x)$ and $B_{i+1,n-1}(x)$ of degree $n-1$, we shall indicate only the first step of a general induction argument. Therefore, we illustrate what happens for the first-degree B-spline $B_{i,1}(x)$.

Now suppose that $x < t_i$. Then also $x < t_{i+1}$, so at x the zero-degree B-splines have the values $B_{i,0}(x) = 0$ and $B_{i+1,0}(x) = 0$. From these values we find, in turn, that $B_{i,1}(x) = 0$.

Now suppose that the knot set is nonrepeating. Let $x = t_i$. Then $B_{i,0}(t_i) = 1$, but $B_{i+1,0}(t_i) = 0$. So

$$B_{i,0}(t_i) = \frac{t_i - t_i}{t_{i+1} - t_i}(1) + \frac{t_{i+2} - t_i}{t_{i+2} - t_{i+1}}(0) = 0$$

Thus $B_{i,1}(x) = 0$ if $x \leq t_i$. Similar arguments show that $B_{i,1}(x) = 0$ if $x \geq t_{i+1}$.

9.4 Let $P_0(0, 0)$, $P_1(1, 2)$, $P_2(2, 1)$, $P_3(3, -1)$, $P_4(4, 10)$, and $P_5(5, 5)$ be given data points. If interpolation based on cubic B-splines is used to find a curve interpolating these data points, find a knot set $t_0, \ldots, t_9$ that can be used to define the cubic B-splines.

SOLUTION

The knot set can be chosen according to one of two schemes. With $m = 3$ and $n = 5$:

1. Choose

$$t_0 = t_1 = t_2 = t_3 = -1 \ (< x_0) \qquad \text{and} \qquad t_6 = t_7 = t_8 = t_9 = 6 \ (> x_n)$$

The remaining knots are chosen according to

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

So

$$t_4 = \frac{1+2+3}{3} = 2 \qquad t_5 = \frac{2+3+4}{3} = 3$$

2. An alternative scheme for cubic splines is

$$t_0 = t_1 = t_2 = t_3 = -1 \qquad t_6 = t_7 = t_8 = t_9 = 6$$

and the remaining knots are chosen according to

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

So

$$t_4 = 2 \qquad t_5 = 3$$

The agreement between knot sets chosen according to these two schemes is a result of the uniform spacing of the data points along the x axis.

9.5 Write the equations that can be used to find an interpolating cubic spline curve to fit the data in Prob. 9.4 using cubic B-spline basis functions.

SOLUTION

With $m = 3$ and $n = 5$, the interpolating spline can be written in terms of cubic B-splines as

$$S_3(x) = \sum_{i=0}^{7} a_i B_{i,3}(x)$$

The interpolation equations for the data points (x_j, y_j), $j = 0, \ldots, 5$ are

$$y_j = S_3(x_j), \qquad j = 0, \ldots, 5$$

With the knot set chosen in Prob. 9.4, the intervals $[t_j, t_{j+4}]$ defined by the knots t_j, $j = 0, \ldots, 9$ are

j	t_j	t_{j+4}
0	−1	2
1	−1	3
2	−1	6
3	−1	6
4	2	6
5	3	6

Because $B_{i,3}(x)$ is nonzero only for $t_i < x < t_{i+4}$, the interpolation equations become

$$\begin{aligned}
y_j &= S_3(x_j) \\
0 &= a_0 B_{0,3}(0) + a_1 B_{1,3}(0) + a_2 B_{2,3}(0) + a_3 B_{3,3}(0) \\
2 &= a_0 B_{0,3}(1) + a_1 B_{1,3}(1) + a_2 B_{2,3}(1) + a_3 B_{3,3}(1) \\
1 &= a_1 B_{1,3}(2) + a_2 B_{2,3}(2) + a_3 B_{3,3}(2) \\
-1 &= a_2 B_{2,3}(3) + a_3 B_{3,3}(3) + a_4 B_{4,3}(3) \\
10 &= a_2 B_{2,3}(4) + a_3 B_{3,3}(4) + a_4 B_{4,3}(4) + a_5 B_{5,3}(4) \\
5 &= a_2 B_{2,3}(5) + a_3 B_{3,3}(5) + a_4 B_{4,3}(5) + a_5 B_{5,3}(5)
\end{aligned}$$

The remaining two equations can be chosen to satisfy prescribed boundary conditions at $x_0 = 0$ and $x_5 = 5$.

9.6 Show that the knot set used in constructing the Bézier–B-spline approximation to a guiding polyline guarantees that the endpoints of the spline coincide with the endpoints of the guiding polyline.

SOLUTION

For an m-degree Bézier–B-spline approximation, the knot set used is

$$\underbrace{0, \ldots, 0}_{m+1}, 1, 2, \ldots, n-m, \underbrace{n-m+1, n-m+1, \ldots, n-m+1}_{m+1}$$

Let $P_0(x_0, y_0, z_0)$ be the first control point of the guiding polyline and $P_n(x_n, y_n, z_n)$ the last. Now $x(t) = \sum_{i=0}^{n} x_i B_{i,m}(t)$ with similar expressions for $y(t)$ and $z(t)$. We wish to show that

$$x(0) = x_0, y(0) = y_0, z(0) = z_0 \quad \text{and} \quad x(n-1) = x_n, y(n-1) = y_n, z(n-1) = z_n$$

Let us restrict ourselves to $m = 2$ (quadratic) B-splines. Then the knot set is

$$\begin{matrix} 0, 0, 0, 1, 2, 3, \ldots, n-2, n-1, n-1, n-1 \\ t_0, t_1, t_2, t_3, t_4, t_5, \ldots, t_{n+1}, t_{n+2}, t_{n+3}, t_{n+4} \end{matrix}$$

Since $B_{i,2}(x)$ is nonzero only over the interval $t_i \le x \le t_{i+3}$, it follows that $B_{i,2}(0)$ is nonzero only if $i = 0$, 1, and 2. So then

$$x(0) = \sum_{i=0}^{n} x_i B_{i,2}(0) = x_0 B_{0,2}(0) + x_1 B_{1,2}(0) + x_2 B_{2,2}(0)$$

To calculate $B_{0,2}$, using the definition and the convention that $\frac{0}{0} = 0$, we obtain

$$B_{0,2}(0) = B_{1,1}(0) \quad \text{and} \quad B_{1,1}(0) = B_{2,0}(0) \quad \text{and} \quad B_{2,0}(0) = 1$$

So $B_{0,2}(0) = 1$ (compare with Prob. 9.3).

To calculate $B_{1,2}(0)$, using the definition

$$B_{1,2}(0) = \frac{2}{2} B_{2,1}(0) \quad \text{and} \quad B_{2,1}(0) = \frac{2}{2-1} B_{3,0}(0) \quad \text{and} \quad B_{3,0}(0) = 0 \quad \text{since} \quad 0 \le t_3 = 1$$

So we have $B_{1,2}(0) = 0$. In a similar manner, we find $B_{2,2}(0) = 0$. Thus, $x(0) = x_0$ and the same for the y and z coordinates.

Similar calculations show that $x(n-1) = x_n$, $y(n-1) = y_n$, and $z(n-1) = z_n$.

9.7 Find the linear Coons surface patch that interpolates the curves of Fig. 9-26.

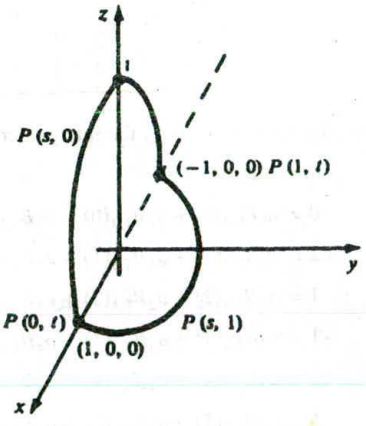


Fig. 9-26

SOLUTION

The four bounding curves can be described parametrically as follows (see Fig. 9-26):

1. $P(s, 0) = (\cos \pi s, 0, \sin \pi s)$ $\quad 0 \le s \le 1$
2. $P(1, t) = (-1, 0, 0)$ $\quad 0 \le t \le 1$
3. $P(s, 1) = (\cos \pi s, \sin \pi s, 0)$ $\quad 0 \le s \le 1$
4. $P(0, t) = (1, 0, 0)$ $\quad 0 \le t \le 1$

Note that curves 2 and 4 are constant curves, that is, points.

The linear Coons surface interpolating these curves can be written as

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

In terms of coordinates:

$$x(s, t) = (\cos \pi s)(1 - t) + (\cos \pi s)t + (1)(1 - s) + (-1)s - (1)(1 - s)(1 - t) - (1)(1 - s)t - (-1)s(1 - t) - (-1)st$$

or

$$x(s, t) = \cos \pi s$$

Now

$$y(s, t) = (0)(1 - t) + (\sin \pi s)t + (0)(1 - s) + (0)s - (0)(1 - s)(1 - t) - (0)(1 - s)t - (0)s(1 - t) - (0)st$$

or

$$y(s, t) = t \sin \pi s$$

Finally

$$z(s, t) = (\sin \pi s)(1 - t) + (0)t + (0)(1 - s) + (0)s - (0)(1 - s)(1 - t) - (0)(1 - s)t - (0)s(1 - t) - (0)st$$

or

$$z(s, t) = (1 - t) \sin \pi s$$

The linear Coons surface is

$$Q(s, t) = [\cos \pi s, t \sin \pi s, (1 - t) \sin \pi s] \qquad 0 \le s, t \le 1$$

9.8 Find the lofting surface defined by linear blending between the cross-section curves in Fig. 9-27.

SOLUTION

The curves in Fig. 9-27 are circles whose equations can be defined parametrically as

$$P_1(s) = (\cos 2\pi s, \sin 2\pi s, 0) \qquad 0 \le s \le 1$$

and

$$P_2(s) = (2 \cos 2\pi s, 2 \sin 2\pi s, 4) \qquad 0 \le s \le 1$$

The lofting surface is then

$$Q(s, t) = (1 - t)P_1(s) + tP_2(s)$$

In terms of coordinates, we find that

$$x(s, t) = (1 - t)(\cos 2\pi s) + t(2 \cos 2\pi s) = (1 + t) \cos 2\pi s$$
$$y(s, t) = (1 - t)(\sin 2\pi s) + t(2 \sin 2\pi s) = (1 + t) \sin 2\pi s$$
$$z(s, t) = (1 - t)(0) + t(4) = 4t$$

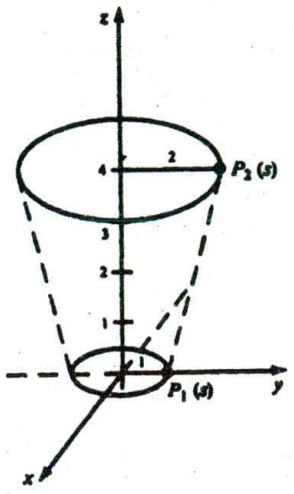


Fig. 9-27

Thus

$$Q(s, t) = [(1 + t)\cos 2\pi s, (1 + t)\sin 2\pi s, 4t] \qquad 0 \le s, t \le 1$$

Supplementary Problems

9.9 Represent a cube using an explicit edge listing.

9.10 Find an explicit representation for linear (degree 1) B-splines in the case of uniformly spaced knots, i.e., $t_{i+1} - t_i = L$.

9.11 For the knot set $t_1 = 1$, $t_2 = 2, \ldots, t_i = 1$, calculate $B_{i,3}$ (5.5).

CHAPTER 10

Hidden Surfaces

Opaque objects that are closer to the eye and in the line of sight of other objects will block those objects or portions of those objects from view. In fact, some surfaces of these opaque objects themselves are not visible because they are eclipsed by the objects' visible parts. The surfaces that are blocked or hidden from view must be "removed" in order to construct a realistic view of the 3D scene (see Fig. 1-3 where only three of the six faces of the cube are shown). The identification and removal of these surfaces is called the *hidden-surface problem*. The solution involves the determination of the closest visible surface along each projection line (Sec. 10.1).

There are many different hidden-surface algorithms. Each can be characterized as either an *image-space method*, in which the pixel grid is used to guide the computational activities that determine visibility at the pixel level (Secs. 10.2, 10.5, and 10.6), or an *object-space method*, in which surface visibility is determined using continuous models in the object space (or its transformation) without involving pixel-based operations (Secs. 10.3 and 10.4).

Notice that the hidden-surface problem has ties to the calculation of shadows. If we place a light source, such as a bulb, at the viewpoint, all surfaces that are visible from the viewpoint are lit directly by the light source and all surfaces that are hidden from the viewpoint are in the shadow of some opaque objects blocking the light.

10.1 DEPTH COMPARISONS

We assume that all coordinates (x, y, z) are described in the normalized viewing coordinate system (Chap. 8).

Any hidden-surface algorithm must determine which edges and surfaces are visible either from the center of projection for perspective projections or along the direction of projection for parallel projections.

The question of visibility reduces to this: given two points $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$, does either point obscure the other? This is answered in two steps:

1. Are P_1 and P_2 on the same projection line?
2. If not, neither point obscures the other. If so, a depth comparison tells us which point is in front of the other.

For an orthographic parallel projection onto the xy plane, P_1 and P_2 are on the same projector if $x_1 = x_2$ and $y_1 = y_2$. In this case, depth comparison reduces to comparing z_1 and z_2. If $z_1 < z_2$, then P_1 obscures P_2 [see Fig. 10-1(a)].

For a perspective projection [see Fig. 10-1(b)], the calculations are more complex (Prob. 10.1). However, this complication can be avoided by transforming all three-dimensional objects so that parallel

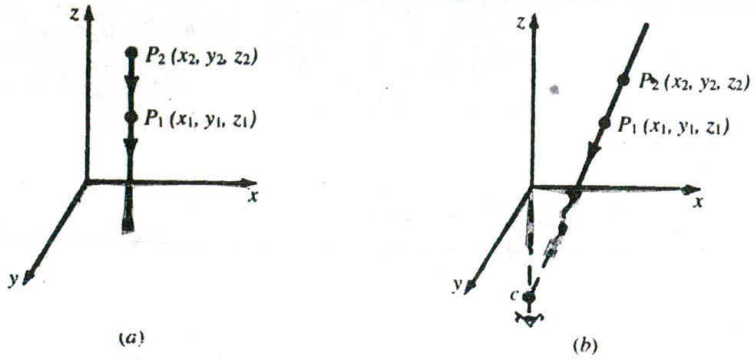


Fig. 10-1

projection of the transformed object is equivalent to a perspective projection of the original object (see Fig. 10-2). This is done with the use of the *perspective to parallel transform* T_p (Prob. 10.2).

If the original object lies in the normalized perspective view volume (Chap. 8), the *normalized perspective to parallel transform*

$$NT_p = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{1-z_f} & \frac{-z_f}{1-z_f} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

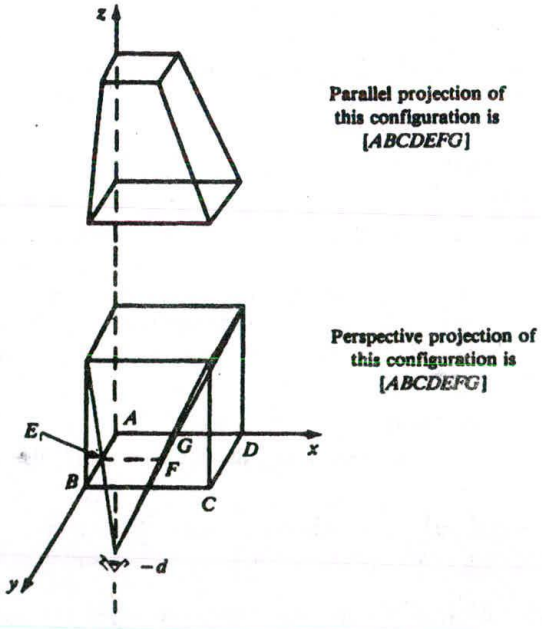


Fig. 10-2

(where z_f is the location of the front clipping plane of the normalized perspective view volume) transforms the normalized perspective view volume into the unit cube bounded by $0 \le x \le 1$, $0 \le y \le 1$, $0 \le z \le 1$ (Prob. 10.3). We call this cube the *normalized display space*. A critical fact is that the normalized perspective to parallel transform preserves lines, planes, and depth relationships.

If our display device has display coordinates $H \times V$, application of the scaling matrix

$$S_{H,V,1} = \begin{pmatrix} H & 0 & 0 & 0 \\ 0 & V & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

transforms the normalized display space $0 \le x \le 1$, $0 \le y \le 1$, $0 \le z \le 1$ onto the region $0 \le x \le H$, $0 \le y \le V$, $0 \le z \le 1$. We cal this region the *display space*. The *display transform* DT_p:

$$DT_p = S_{H,V,1} \cdot NT_p$$

transforms the normalized perspective view volume onto the display space.

Clipping must be done against the normalized perspective view volume prior to applying the transform NT_p. An alternative to this is to combine NT_p with the normalizing transformation N_{per} (Chap. 8), forming the single transformation $NT'_p = NT_p \cdot N_{\text{per}}$. Then clipping is done in homogeneous coordinate space. This method for performing clipping is not covered in this book.

We now describe several algorithms for removing hidden surfaces from scenes containing objects defined with planar (i.e., flat), polygonal faces. We assume that the displays transform DT_p has been applied (if a perspective projection is being used), so that we always deal with parallel projections in display space.

10.2 Z-BUFFER ALGORITHM

We say that a point in display space is "seen" from pixel (x, y) if the projection of the point is scan-converted to this pixel (Chap. 3). The Z-buffer algorithm essentially keeps track of the smallest z coordinate (also called the *depth value*) of those points which are seen from pixel (x, y). These Z values are stored in what is called the Z buffer.

Let $Z_{\text{buf}}(x, y)$ denote the current depth value that is stored in the Z buffer at pixel (x, y). We work with the (already) projected polygons P of the scene to be rendered.

The Z-buffer algorithm consists of the following steps.

1. Initialize the screen to a background color. Initialize the Z buffer to the depth of the back clipping plane. That is, set

$$Z_{\text{buf}}(x, y) = Z_{\text{back}}, \qquad \text{for every pixel } (x, y)$$

2. Scan-convert each (projected) polygon P in the scene (Chap. 3) and during this scan-conversion process, for each pixel (x, y) that lies inside the polygon:

   (*a*) Calculate $Z(x, y)$, the depth of the polygon at pixel (x, y).

   (*b*) If $Z(x, y) < Z_{\text{buf}}(x, y)$, set $Z_{\text{buf}}(x, y) = Z(x, y)$ and set the pixel value at (x, y) to the color of the polygon P at (x, y). In Fig. 10-3, points P_1 and P_2 are both scan-converted to pixel (x, y); however, since $z_1 < z_2$, P_1 will obscure P_2 and the P_1 z value, z_1, will be stored in the Z buffer.

Although the Z-buffer algorithm requires Z-buffer memory storage proportional to the number of pixels on the screen, it does not require additional memory for storing all the objects comprising the scene. In fact, since the algorithm processes polygons one at a time, the total number of objects in a scene can be arbitrarily large.

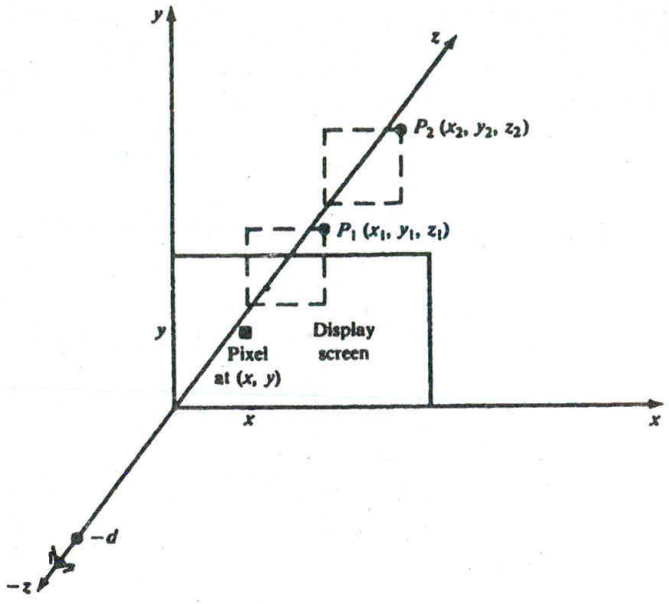


Fig. 10-3

10.3 BACK-FACE REMOVAL

Object surfaces that are orientated away from the viewer are called *back-faces*. The back-faces of an opaque polyhedron are completely blocked by the polyhedron itself and hidden from view. We can therefore identify and remove these back-faces based solely on their orientation without further processing (projection and scan-conversion) and without regard to other surfaces and objects in the scene.

Let $\mathbf{N} = (A, B, C)$ be the normal vector of a planar polygonal face, with $\mathbf{N}$ pointing in the direction the polygon is facing. Since the direction of viewing is the direction of the positive z axis (see Fig. 10-3), the polygon is facing away from the viewer when $C > 0$ (the angle between $\mathbf{N}$ and the z axis is less than 90°). The polygon is also classified as a back-face when $C = 0$, since in this case it is parallel to the line of sight and its projection is either hidden or overlapped by the edge(s) of some visible polygon(s).

Although this method identifies and removes back-faces quickly it does not handle polygons that face the viewer but are hidden (partially or completely) behind other surfaces. It can be used as a preprocessing step for other algorithms.

10.4 THE PAINTER'S ALGORITHM

Also called the *depth sort* or *priority algorithm*, the painter's algorithm processes polygons as if they were being painted onto the view plane in the order of their distance from the viewer. More distance polygons are painted first. Nearer polygons are painted on or over more distance polygons, partially or totally obscuring them from view. The key to implementing this concept is to find a priority ordering of the polygons in order to determine which polygons are to be painted (i.e., scan-converted) first.

Any attempt at a priority ordering based on depth sorting alone results in ambiguities that must be resolved in order to correctly assign priorities. For example, when two polygons overlap, how do we decide which one obscures the other? (See Fig. 10-4.)

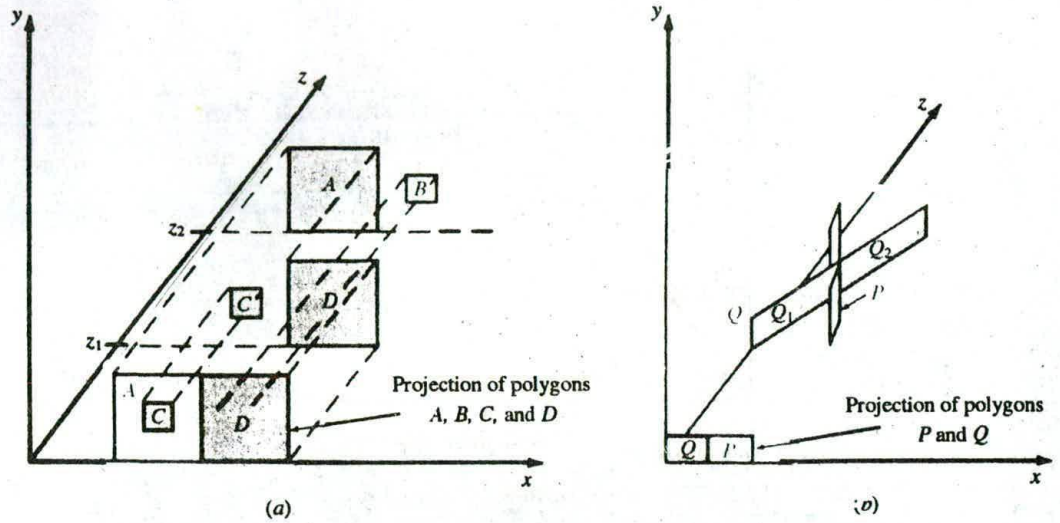


Fig. 10-4 Projection of opaque polygons.

Assigning Priorities

We assign priorities to polygons by determining if a given polygon P obscures other polygons. If the answer is no, then P should be painted first. Hence the key test is to determine whether polygon P does *not* obscure polygon Q.

The z extent of a polygon is the region between the planes $z = z_{min}$ and $z = z_{max}$ (Fig. 10-5). Here, z_{min} is the smallest of the z coordinates of all the polygon's vertices, and z_{max} is the largest.

Similar definitions hold for the x and y extents of a polygon. The intersection of the x, y, and z extents is called the *extent*, or bounding box, of the polygon.

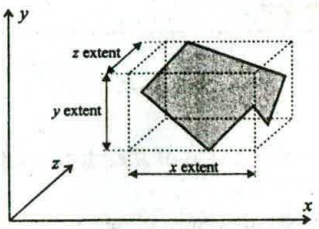


Fig. 10-5

Testing Whether P Obscures Q

Polygon P does not obscure polygon Q if any one of the following tests, applied in sequential order, is true.

Test 0: the z extents of P and Q do not overlap and $z_{Q_{max}}$ of Q is smaller than $z_{P_{min}}$ of P. Refer to Fig. 10-6.

Test 1: the y extents of P and Q do not overlap. Refer to Fig. 10-7.

Test 2: the x extents of P and Q do not overlap.

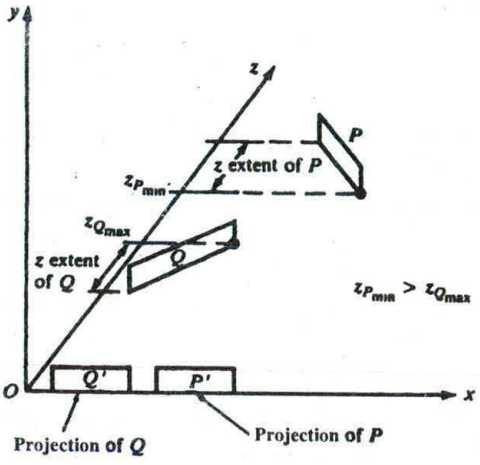


Fig. 10-6

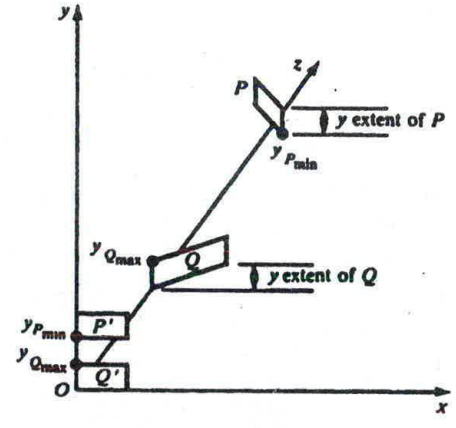


Fig. 10-7

Test 3: all the vertices of P lie on that side of the plane containing Q which is farthest from the viewpoint. Refer to Fig. 10-8.

Test 4: all the vertices of Q lie on that side of the plane containing P which is closest to the viewpoint. Refer to Fig. 10-9.

Test 5: the projections of the polygons P and Q onto the view plane do not overlap. This is checked by comparing each edge of one polygon against each edge of the other polygon to search for intersections.

The Algorithm

1. Sort all polygons into a polygon list according to z_{max} (the largest z coordinate of each polygon's vertices). Starting from the end of the list, assign priorities for each polygon P, in order, as described in steps 2 and 3 (below).

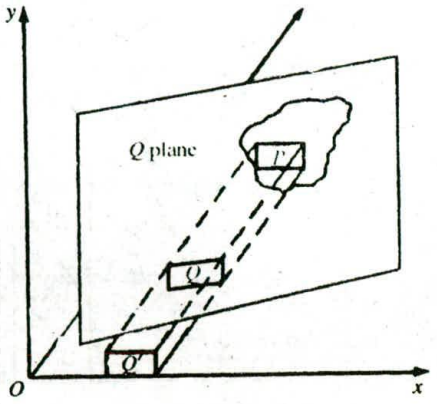


Fig. 10-8

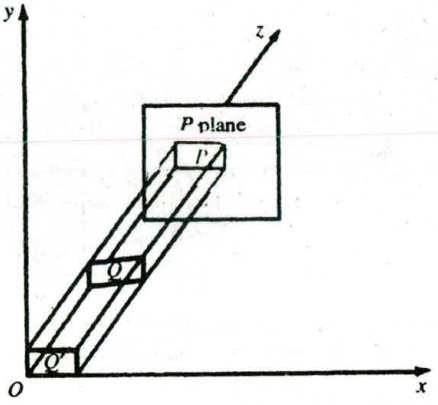


Fig. 10-9

2. Find all polygons Q (preceding P) in the polygon list whose z extents overlap that of P (test 0).
3. For each Q, perform tests 1 through 5 until true.
   (*a*) If every Q passes, scan-convert polygon P.
   (*b*) If false for some Q, swap P and Q on the list. Tag Q as swapped. If Q has already been tagged, use the plane containing polygon P to divide polygon Q into two polygons. Q_1 and Q_2 [see Fig. 10-4(*b*)]. The polygon-clipping techniques described in Chap. 5 can be used to perform the division. Remove Q from the list and place Q_1 and Q_2 on the list, in sorted order.

Sometimes the polygons are subdivided into triangles before processing, thus reducing the computational effort for polygon subdivision in step 3.

10.5 SCAN-LINE ALGORITHM

A scan-line algorithm consists essentially of two nested loops, an x-scan loop nested within a y-scan loop.

y Scan

For each y value, say, $y = \alpha$, intersect the polygons to be rendered with the scan plane $y = \alpha$. This scan plane is parallel to the xz plane, and the resulting intersections are line segments in this plane (see Fig. 10-10).

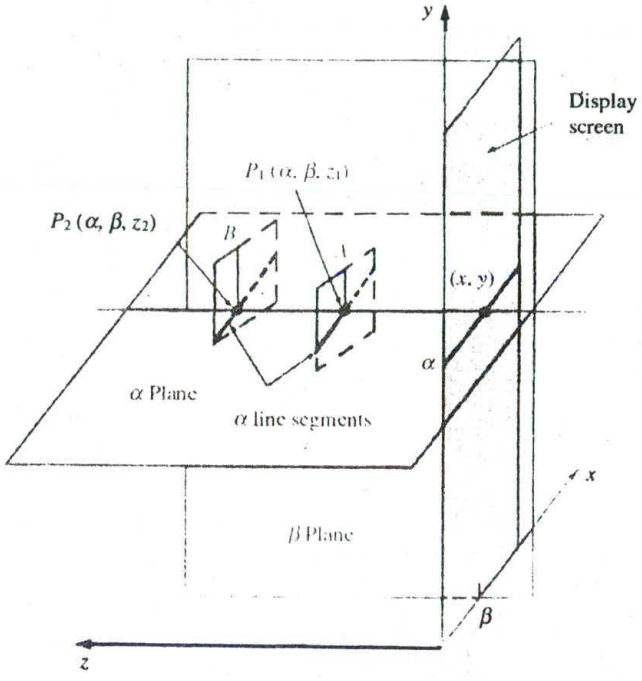


Fig. 10-10

x Scan

1. For each x value, say, $x = \beta$, intersect the line segments found above with the x-scan line $x = \beta$ lying on the y-scan plane. This intersection results in a set of points that lies on the x-scan line.
2. Sort these points with respect to their z coordinates. The point (x, y, z) with the smallest z value is visible, and the color of the polygon containing this point is the color set at the pixel corresponding to this point.

In order to reduce the amount of calculation in each scan-line loop, we try to take advantage of relationships and dependencies, called *coherences*, between different elements that comprise a scene.

Types of Coherence

1. *Scan-line coherence.* If a pixel on a scan line lies within a polygon, pixels near it will most likely lie within the polygon.
2. *Edge coherence.* If an edge of a polygon intersects a given scan line, it will mot likely intersect scan lines near the given one.

3. *Area coherence.* A small area of an image will most likely lie within a single polygon.
4. *Spatial coherence.* Certain properties of an object can be determined by examining the *extent* of the object, that is, a geometric figure which circumscribes the given object. Usually the extent is a rectangle or rectangle solid (also called a *bounding box*).

Scan-line coherence and edge coherence are both used to advantage in scan-converting polygons (Chap. 3).

Spatial coherence is often used as a preprocessing step. For example, when determining whether polygons intersect, we can eliminate those polygons that don't intersect by finding the rectangular extent of each polygon and checking whether the extents intersect—a much simpler problem (see Fig. 10-11). [*Note*: In Fig. 10-11 objects A and B do not intersect; however, objects A and C, and B and C, do intersect. In preprocessing, corner points would be compared to determine whether there is an intersection. For example, the edge of object A is at coordinate $P_3 = (6, 4)$ and the edge of object B is at coordinate $P_4 = (7, 3)$.] Of course, even if the extents intersect, this does not guarantee that the polygons intersect. See Fig. 10-12 and note that the extents of A' and B' overlap even though the polygons do not.

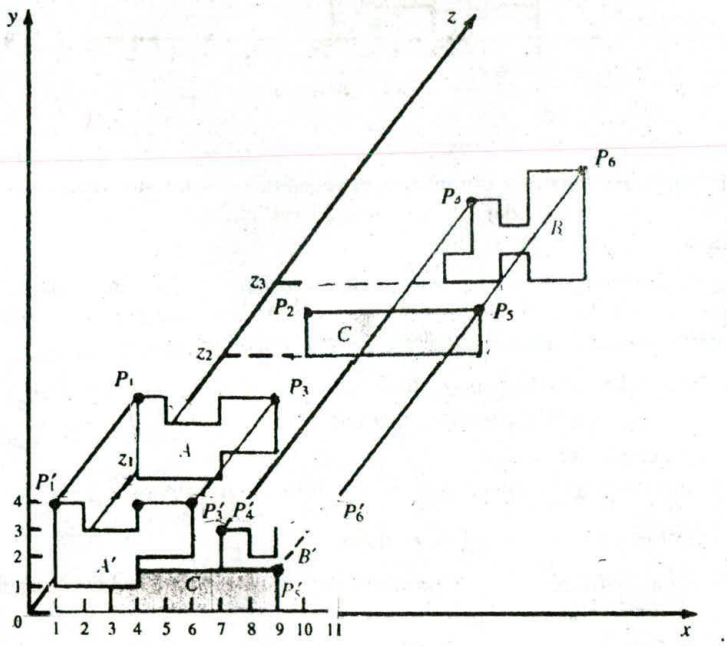


Fig. 10-11

Coherences can simplify calculations by making them incremental, as opposed to absolute. This is illustrated in Prob. 10.13.

A Scan-line Algorithm

In the following algorithm, scan line and edge coherence are used to enhance the processing done in the y-scan loop as follows. Since the y-scan loop constructs a list of potentially visible line segments, instead of reconstructing this list each time the y-scan line changes (absolute calculation), we keep the list and update it according to how it has changed (incremental calculation). This processing is facilitated by

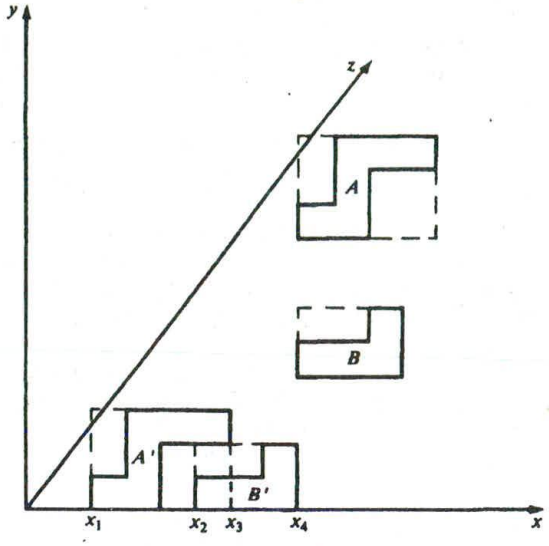


Fig. 10-12

the use of what is called the *edge list*, and its efficient construction and maintenance is at the heart of the algorithm (see Chap. 3, Sec. 3.7, under "A Scan-line Algorithm").

The following data structures are created:

1. *The edge list*—contains all nonhorizontal edges (horizontal edges are automatically displayed) of the projections of the polygons in the scene. The edges are sorted by the edge's smaller y coordinate ($y_{\min}$). Each edge entry in the edge list also contains:
   (*a*) The x coordinate of the end of the edge with the smaller y coordinate.
   (*b*) The y coordinate of the edge's other end ($y_{\max}$).
   (*c*) The increment $\Delta x = 1/m$.
   (*d*) A pointer indicating the polygon to which the edge belongs.
2. *The polygon list*—for each polygon, contains
   (*a*) The equation of the plane within which the polygon lies—used for depth determination, i.e., to find the z value at pixel (x, y).
   (*b*) An IN/OUT flag, initialized to OUT (this flag is set depending on whether a given scan line is in or out of the polygon).
   (*c*) Color information for the polygon.

The algorithm proceeds as follows.

I. *Initialization.*
   (*a*) Initialize each screen pixel to a background color.
   (*b*) Set y to the smallest $y_{\min}$ value in the edge list.

   Repeat steps II and III (below) until no further processing can be performed.

II. *y-scan loop.* Activate edges whose $y_{\min}$ is equal to y. Sort active edges in order of increasing x.

III. *x-scan loop.* Process, from left to right, each active edge as follows:

(*a*) Invert the IN/OUT flag of the polygon in the polygon list which contains the edge. Count the number of active polygons whose IN/OUT flag is set to IN. If this number is 1, only one polygon is visible. All pixel values from this edge and up to the next edge are set to the color of the polygon. If this number is greater than 1, determine the visible polygon by the smallest z value of each polygon at the pixel under consideration. These z values are found from the equation of the plane containing the polygon. The pixels from this edge and up to the next edge are set to the color of this polygon, unless the polygon becomes obscured by another before the next edge is reached, in which case we set the remaining pixels to the color of the obscuring polygon. If this number is 0, pixels from this edge and up to the next one are left unchanged.

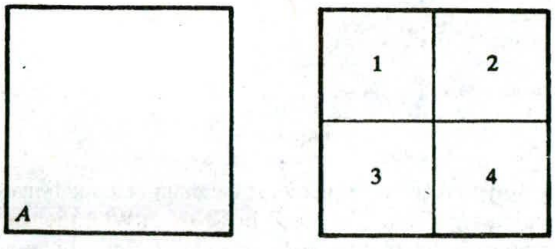
(*b*) When the last active edge is processed, we then proceed as follows:

1. Remove those edges for which the value of y_{max} equals the present scan-line value y. If no edges remain, the algorithm has finished.
2. For each remaining active edge, in order, replace x by $x + 1/m$. This is the edge intersection with the next scan line $y + 1$ (see Prob. 10.13).
3. Increment y to $y + 1$, the next scan line, and repeat step II.

10.6 SUBDIVISION ALGORITHM

The subdivision algorithm is a recursive procedure based on a two-step strategy that first decides which projected polygons overlap a given area A on the screen and are therefore potentially visible in that area. Second, in each area these polygons are further tested to determine which ones will be visible within this area and should therefore be displayed. If a visibility decision cannot be made, this screen area, usually a rectangular window, is further subdivided either until a visibility decision can be made, or until the screen area is a single pixel.

Starting with the full screen as the initial area, the algorithm divides an area at each stage into four smaller areas, thereby generating a quad tree (see Fig. 10-13).



(*a*) Initial area. (*b*) First subdivision.

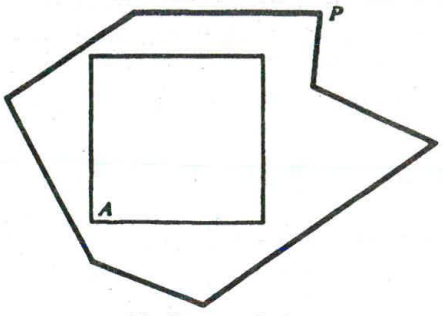
1	2	1	2
3	4	3	4
1	2	1	2
3	4	3	4

1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4

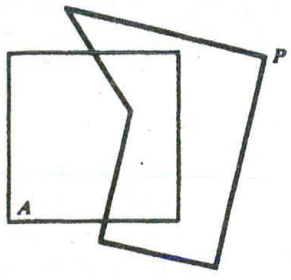
(*c*) Second subdivision. (*d*) Third subdivision.

Fig. 10-13

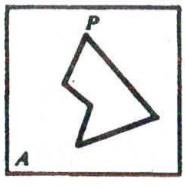
The processing exploits area coherence by classifying polygons P with respect to a given screen area A into the following categories: (1) *surrounding polygon*—polygon that completely contains the area [Fig. 10-14(*a*)], (2) *interesecting polygon*—polygon that intersects the area [Fig. 10-14(*b*)], (3) *contained polygon*—polygon that is completely contained within the area [Fig. 10-14(*c*)], and (4) *disjoint polygon*—polygon that is disjoint from the area [Fig. 10-14(*d*)].



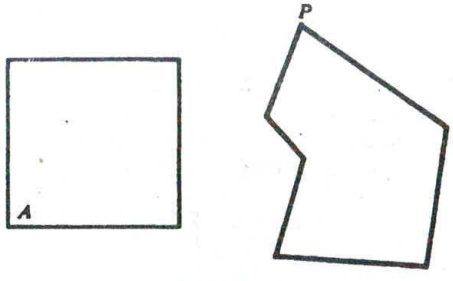
(*a*) *P* surrounds *A*.



(*b*) *P* intersects *A*.



(*c*) *P* is contained in *A*.



(*d*) *P* is disjoint from *A*.

Fig. 10-14

The classification of the polygons within a picture is the main computational expense of the algorithm and is analogous to the clipping algorithms discussed in Chap. 5. With the use of one of these clipping algorithms, a polygon in category 2 (intersecting polygon) can be clipped into a contained polygon and a disjoint polygon (see Fig. 10-15). Therefore, we could proceed as if category 2 were eliminated.

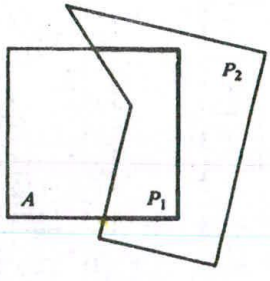


Fig. 10-15

For a given screen area, we keep a potentially visible polygons list (PVPL), those in categories 1, 2, and 3. (Disjoint polygons are clearly not visible.) Also, note that on subdivision of a screen area, surrounding and disjoint polygons remain surrounding and disjoint polygons of the newly formed areas. Therefore, only contained and intersecting polygons need to be reclassified.

Removing Polygons Hidden by a Surrounding Polygon

The key to efficient visibility computation lies in the fact that a polygon is not visible if it is in back of a surrounding polygon. Therefore, it can be removed from the PVPL. To facilitate processing, this list is sorted by z_{min}, the smallest z coordinate of the polygon within this area. In addition, for each surrounding polygon S, we also record its largest z coordinate, $z_{S_{max}}$.

If, for a polygon P on the list, $z_{P_{min}} > z_{S_{max}}$ (for a surrounding polygon S), then P is hidden by S and thus is not visible. In addition, all other polygons after P on the list will also be hidden by S, so we can remove these polygons from the PVPL.

Subdivision Algorithm

1. Initialize the area to be the whole screen.
2. Create a PVPL with respect to an area, sorted on z_{min} (the smallest z coordinate of the polygon within the area). Place the polygons in their appropriate categories. Remove polygons hidden by a surrounding polygon and remove disjoint polygons.
3. Perform the visibility decision tests:
   (*a*) If the list is empty, set all pixels to the background color.
   (*b*) If there is exactly one polygon in the list and it is a classified as intersecting (category 2) or contained (category 3), color (scan-convert) the polygon, and color the remaining area to the background color.
   (*c*) If there is exactly one polygon on the list and it is a surrounding one, color the area the color of the surrounding polygon.
   (*d*) If the area is the pixel (x, y), and neither *a*, *b*, nor *c* applies, compute the z coordinate $z(x, y)$ at pixel (x, y) of all polygons on the PVPL. The pixel is then set to the color of the polygon with the smallest z coordinate.
4. If none of the above cases has occurred, subdivide the screen area into fourths. For each area, go to step 2.

10.7 HIDDEN-LINE ELIMINATION

Although there are special-purpose hidden-line algorithms, each of the above algorithms can be modified to eliminate hidden lines or edges. This is especially useful for wireframe polygonal models where the polygons are unfilled. The idea is to use a color rule which fills all the polygons with the background color—say, black—and the edges and lines a different color—say, white. The use of a hidden-surface algorithm now becomes a hidden-line algorithm.

10.8 THE RENDERING OF MATHEMATICAL SURFACES

In plotting a mathematical surface described by an equation $z = F(x, y)$, where $x_{min} \le x \le x_{max}$ and $y_{min} \le y \le y_{max}$, we could use any of the hidden-surface algorithms so far described. However, these general algorithms are inefficient when compared to specialized algorithms that take advantage of the structure of this type of surface.

The mathematical surface is rendered as a wireframe model by drawing both the *x-constant curves* $z = F(\text{const}, y)$ and the *y-constant curves* $z = F(x, \text{const})$ (see Fig. 10-16). Each such curve is rendered as

a polyline, where the illusion of smoothness is achieved by using a fine resolution (i.e., short line segments) in drawing the polyline (Chap. 3).

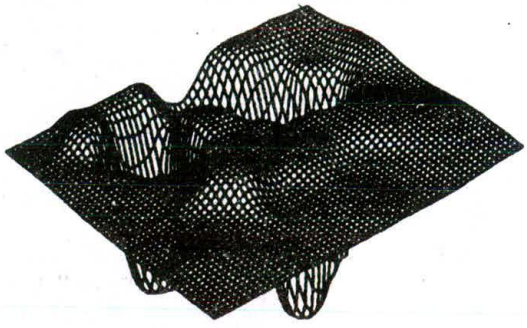


Fig. 10-16

Choose an $M \times N$ plotting resolution

$$x_{\min} \le x_1 < x_2 < \cdots < x_M \le x_{\max} \qquad \text{and} \qquad y_{\min} \le y_1 < y_2 < \cdots < y_N \le y_{\max}$$

The corresponding z values are $z_{ij} = F(x_i, y_j)$. An x-constant polyline, say $x = x_j$, has vertices

$$P_1(x_j, y_1), \ldots, P_N(x_j, y_N)$$

Similarly, the $y = y_k$ polyline has vertices

$$Q_1(x_1, y_k), \ldots, Q_M(x_M, y_k)$$

Choosing a view plane and a center of projection or viewpoint $C(a, b, c)$, we create a perspective view of the surface onto this view plane by using the transformations developed in Chap. 7. So a point $[x, y, F(x, y)]$ on the surface projects to a point (p, q) in view plane coordinates. By applying an appropriate 2D viewing transformation (Chap. 5), we can suppose that p and q line within the horizontal and vertical plotting dimensions of the plotting device, say, $H \times V$ pixels.

The Perimeter Method for Rendering the Surface

Each plotted x and y constant polyline outlines a polygonal region on the plotting screen (Fig. 10-17).

The algorithm is based on the following observations: (1) *ordering*—the x- and y-constant curves (i.e., polylines) are drawn in order starting with the one closest to the viewpoint and (2) *visibility*—we draw only that part of the polyline that is outside the perimeter of all previously drawn regions (Fig. 10-18). One implementation of the visibility condition uses a min–max array A, of length H (that of the plotting device), which contains, at each horizontal pixel position i, the maximum (and/or minimum) vertical pixel value

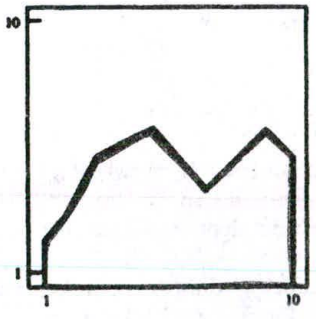


Fig. 10-17

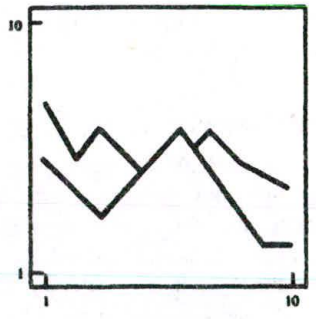


Fig. 10-18

drawn thus far at i; that is $A(i) = \frac{\max}{\min}$ (vertical pixel values drawn so far at i) (Fig. 10-19). Selection of the max results in a drawing of the top of the surface. The min is used to render the bottom of the surface, and the max and min yields both top and bottom.

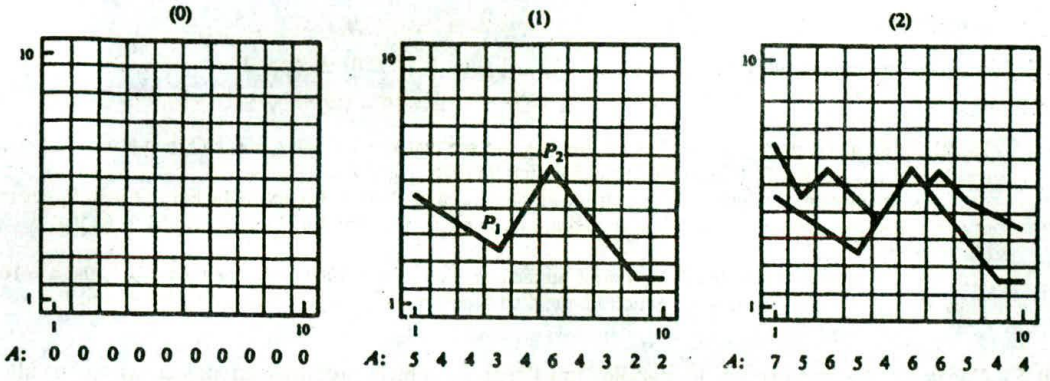


Fig. 10-19 Updating min–max array A (using maximum values only).

The Visibility Test

Suppose that (p', q') is the pixel that corresponds to the point (p, q). Then this pixel is *visible* if either

$$q' > A(p') \quad \text{or} \quad q' < A(p')$$

where A is the min–max array.

The visibility criteria for a line segment are: (1) the line segment is visible if both its endpoints are visible; (2) the line segment is invisible if both its endpoints are not visible; and (3) if only one endpoint is visible, the min–max array is tested to find the visible part of the line.

Drawing the x- or y-constant polylines thus consists of testing for the visibility of each line segment and updating the min–max array as necessary. Since a line segment will, in general, span several horizontal pixel positions (see segment $\overline{P_1P_2}$ in Fig. 10-19), the computation of $A(i)$ for these intermediate pixels is found by using the slope of the line segment or by using Bresenham's method described in Chap. 3.

The Wright Algorithm for Rendering Mathematical Surfaces

The drawing of the surface $z = F(x, y)$ proceeds as follows.

1. To perform initialization, determine whether the viewpoint is closer to the x or the y axis. Suppose that it is closer to the x axis. We next locate the x-constant curve that is closest to the viewpoint at $x = 1$.
   (*a*) Initialize the min–max array to some base value, say, zero.
   (*b*) Start with the x-constant curve found above.
2. Repeat the following steps using the visibility test for drawing line segments and updating the min–max array each time a line segment is drawn:
   (*a*) Draw the x-constant polyline.
   (*b*) Draw those parts of each y-constant polyline that lie between the previously drawn x-constant polyline and the next one to be drawn.
   (*c*) Proceed, in the direction of increasing x, to the next x-constant polyline.

Solved Problems

10.1 Given points $P_1(1, 2, 0)$, $P_2(3, 6, 20)$, and $P_3(2, 4, 6)$ and a viewpoint $C(0, 0, -10)$, determine which points obscure the others when viewed from C.

SOLUTION

The line joining the viewpoint $C(0, 0, -10)$ and point $P_1(1, 2, 0)$ is (App. 2)

$$x = t \qquad y = 2t \qquad z = -10 + 10t$$

To determine whether $P_2(3, 6, 20)$ lies on this line, we see that $x = 3$ when $t = 3$, and then at $t = 3$, $x = 3$, $y = 6$, and $z = 20$. So P_2 lies on the projection line through C and P_1.

Next we determine which point is in front with respect to C. Now C occurs on the line at $t = 0$, P_1 occurs at $t = 1$, and P_2 occurs at $t = 3$. Thus comapring t values, P_1 is in front of P_2 with respect to C; that is, P_1 obscures P_2.

We now determine whether $P_3(2, 4, 6)$ is on the line. Now $x = 2$ when $t = 2$ and then $y = 4$, and $z = 10$. Thus $P_3(2, 4, 6)$ is not on this projection line and so it nether obscures nor is obscured by P_1 and P_2.

10.2 Construct the perspective to parallel transform T_p which produces an object whose parallel projection onto the xy plane yields the same image as the perspective projection of the original object onto the normalized view plane $z = c_z'/(c_z' + b)$ (Chap. 8, Prob. 8.6) with respect to the origin as the center of projection.

SOLUTION

The perspective projection onto the plane $z = c_z'/(c_z' + b)$ with respect to the origin is (Chap. 7, Prob. 7.4):

$$Per = \begin{pmatrix} z_v & 0 & 0 & 0 \\ 0 & z_v & 0 & 0 \\ 0 & 0 & z_v & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

where $z_v = c_z'/(c_z' + b)$. The perspective projection onto the view plane of a point $P(x, y, z)$ is the point

$$P'\left(\frac{z_v x}{z}, \frac{z_v y}{z}, z_v\right)$$

Define the perspective to parallel transform T_p to be

$$T_p = \begin{pmatrix} z_v & 0 & 0 & 0 \\ 0 & z_v & 0 & 0 \\ 0 & 0 & \dfrac{1}{1 - z_f} & \dfrac{-z_f}{1 - z_f} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

(where $z = z_f$ is the location of the normalized front clipping plane; see Chap. 8, Prob. 8.6).

Now, applying the perspective to parallel transform T_p to the point $P(x, y, z)$, we produce the point

$$Q'\left(\frac{z_v x}{z}, \frac{z_v y}{z}, \frac{z - z_f}{z(1 - z_f)}\right)$$

The parallel projection of Q' onto the xy plane produces the point

$$Q'\left(\frac{z_v x}{z}, \frac{z_v y}{z}, 0\right)$$

So Q' and P' produce the same projective image. Furthermore, T_p transforms the normalized perspective view volume bounded by $x = z$, $x = -z$, $y = z$, $y = -z$, $z = z_f$, and $z = 1$ to the rectangular volume bounded by $x = z_v$, $x = -z_v$, $y = z_v$, $y = -z_v$, $z = 0$, and $z = 1$.

10.3 Show that the normalized perspective to parallel transform NT_p preserves the relationships of the original perspective transformation while transforming the normalized perspective view volume into the unit cube.

SOLUTION

From Prob. 10.2, the perspective to parallel transform T_p transforms a point $P(x, y, z)$ to a point

$$Q\left(\frac{z_v x}{z}, \frac{z_v y}{z}, \frac{z - z_f}{z(1 - z_f)}\right)$$

The image under parallel projection of this point onto the xy plane is

$$Q'\left(\frac{z_v x}{z}, \frac{z_v y}{z}, 0\right)$$

The factor z_v can be set equal to 1 without changing the relation between points Q and Q'.

The matrix that transforms $P(x, y, z)$ to the point $Q\left(\frac{x}{z}, \frac{y}{z}, \frac{z - z_f}{z(1 - z_f)}\right)$ is then

$$\bar{T}_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1 - z_f} & \dfrac{-z_f}{1 - z_f} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

In addition, this matrix transforms the normalized perspective view volume to the rectangular view volume bounded by $x = 1$, $x = -1$, $y = 1$, $y = -1$, $z = 0$, and $z = 1$.

We next translate this view volume so that the corner point $(-1, -1, 0)$ translates to the origin. The translation matrix that does this is

$$T_{(1,1,0)} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The new region is a volume bounded by $x = 0$, $x = 2$, $y = 0$, $y = 2$, $z = 0$, and $z = 1$.

Finally, we scale in the x and y direction by a factor $\frac{1}{2}$ so that the final view volume is the unit cube: $x = 0$, $x = 1$, $y = 0$, $y = 1$, $z = 0$, and $z = 1$. The scaling matrix is

$$S_{1/2,1/2,1} = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The final normalized perspective to parallel transform is

$$NT_p = S_{1/2,1/2,1} \cdot T_{(1,1,0)} \cdot \bar{T}_p = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \dfrac{1}{1 - z_f} & \dfrac{-z_f}{1 - z_f} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

10.4 Why are hidden-surface algorithms needed?

SOLUTION

Hidden-surface algorithms are needed to determine which objects and surface will obscure those objects and surfaces that are in back of them, thus rendering a more realistic image.

10.5 What two steps are required to determine whether any given points $P_1(x_1, y_1, z_1)$ obscures another point $P_2(x_2, y_2, z_2)$? (See Fig. 10-1.)

SOLUTION

It must be determined (1) whether the two points lie on the same projection line and (2) if they do, which point is in front of the other.

10.6 Why is it easier to locate hidden surfaces when parallel projection is used?

SOLUTION

There are no vanishing points in parallel projection. As a result, any point $P(a, b, z)$ will line on the same projector as any other point having the same x and y coordinates (a, b). Thus only the z component must be compared to determine which point is closest to the viewer.

10.7 How does the Z-buffer algorithm determine which surfaces are hidden?

SOLUTION

The Z-buffer algorithm sets up a two-dimensional array which is like the frame buffer; however the Z buffer stores the depth value at each pixel rather than the color, which is stored in the frame buffer. By setting the initial values of the Z buffer to some large number, usually the distance of back clipping plane, the problem of determining which surfaces are closer is reduced to simply comparing the present depth values stored in the Z buffer at pixel (x, y) with the newly calculated depth value at pixel (x, y). If this new value is less than the present Z-buffer value (i.e., closer along the line of sight), this value replaces the present value and the pixel color is changed to the color of the new surface.

10.8 Using a 2×2 pixel display, show how the Z-buffer algorithm would determine the color of each pixel for the given objects A and B in Fig. 10-20.

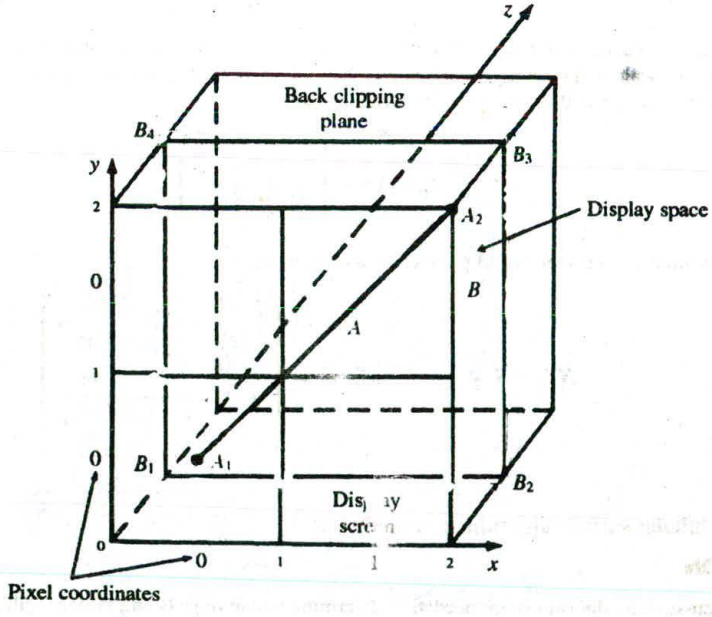


Fig. 10-20

SOLUTION

The display space for the 2×2 pixel display is the region $0 \le x \le 2$, $0 \le y \le 2$, and $0 \le z \le 1$. In Fig. 10-20, A is the line with display space coordinates $A_1(\frac{1}{2}, \frac{1}{2}, 0)$ and $A_2(2, 2, 0)$; line A is on the display screen in front of square B. B is the square with display space coordinates $B_1(0, 0, \frac{1}{2})$, $B_2(2, 0, \frac{1}{2})$, $B_3(2, 2, \frac{1}{2})$, and $B_4(0, 2, \frac{1}{2})$. The displayed image of A (after projection and scan conversion) would appear on a 2×2 pixel display as

1	y	a
0	a	y
	0	1

where a is the color of A and y is the background color. We have used the fact (Chap. 3, Sec. 3.1) that a point (x, y) scan-converts to the pixel [Floor(x), Floor(y)]. (We assume for consistency that $2 \equiv 1.99\ldots$.) The displayed image of B is

1	b	b
0	b	b
	0	1

where b is the color of B. We apply the Z-buffer algorithm to the picture composed of objects A and B as follows:

1. Perform initialization. The Z buffer is set equal to the depth of the back clipping plane $z = 1$, and the frame buffer is initialized to a background color y.

Frame buffer =

1	y	y
0	y	y
	0	1

Z buffer =

1	1	1
0	1	1
	0	1

2. Apply the algorithm to object A.
   (*a*) The present Z-buffer value at pixel (0, 0) is that of the back clipping plane, i.e., $Z_{buf}(0, 0) = 1$. The depth vapue of A at pixel (0, 0) is $z = 0$. Then $Z_{buf}(0, 0)$ is changed to 0 and pixel (0, 0) has the color of A.

Frame buffer =

1	y	y
0	a	y
	0	1

Z buffer =

1	1	1
0	0	1
	0	1

   (*b*) Object A is not seen from pixel (1, 0), so the Z-buffer value is unchanged.

Frame =

1	y	y
0	a	y
	0	1

Z_{buf} =

1	1	1
0	0	1
	0	1

   (*c*) Object A is not seen from pixel (0, 1), so the Z-buffer value is unchanged.

Frame =

1	y	y
0	a	y
	0	1

Z_{buf} =

1	1	1
0	0	1
	0	1

(*d*) The depth value of A at pixel (1, 1) is 0. Since this is less than the present Z-buffer value of 1, pixel (1, 1) takes the color of A.

$$\text{Frame} = \begin{array}{c|cc} 1 & y & a \\ 0 & a & y \\ & 0 & 1 \end{array} \qquad Z_{\text{buf}} = \begin{array}{c|cc} 1 & 1 & 0 \\ 0 & 0 & 1 \\ & 0 & 1 \end{array}$$

3. Apply the algorithm to B.

(*a*) The depth value for B at pixel (0, 0) is $\frac{1}{2}$, and $Z_{\text{buf}}(0, 0) = 0$. So the color of pixel (0, 0) is unchanged.

$$\text{Frame} = \begin{array}{c|cc} 1 & y & a \\ 0 & a & y \\ & 0 & 1 \end{array} \qquad Z_{\text{buf}} = \begin{array}{c|cc} 1 & 1 & 0 \\ 0 & 0 & 1 \\ & 0 & 1 \end{array}$$

(*b*) The depth value of B at pixel (1, 0) is $\frac{1}{2}$. The present Z-buffer value is 1. So the Z-buffer value is set to $\frac{1}{2}$ and pixel (1, 0) takes the color of B.

$$\text{Frame} = \begin{array}{c|cc} 1 & y & a \\ 0 & a & b \\ & 0 & 1 \end{array} \qquad Z_{\text{buf}} = \begin{array}{c|cc} 1 & 1 & 0 \\ 0 & 0 & \frac{1}{2} \\ & 0 & 1 \end{array}$$

(*c*) The depth value of B at pixel (0, 1) is $\frac{1}{2}$. The present Z-buffer value is 1, so the color at pixel (0, 1) is set to that of B, and the Z buffer is updated.

$$\text{Frame} = \begin{array}{c|cc} 1 & b & a \\ 0 & a & b \\ & 0 & 1 \end{array} \qquad Z_{\text{buf}} = \begin{array}{c|cc} 1 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \\ & 0 & 1 \end{array}$$

(*d*) The depth value of B at pixel (1, 1) is $\frac{1}{2}$. The present Z-buffer value is 0. So the color at pixel (1, 1) remains unchanged.

$$\text{Frame} = \begin{array}{c|cc} 1 & b & a \\ 0 & a & b \\ & 0 & 1 \end{array} \qquad Z_{\text{buf}} = \begin{array}{c|cc} 1 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \\ & 0 & 1 \end{array}$$

The final form of the Z buffer indicates that line A lines in front of B.

10.9 What is the maximum number of objects that can be presented by using the Z-buffer algorithm?

SOLUTION

The total number of objects that can be handled by the Z-buffer algorithm is arbitrary because each object is processed one at a time.

10.10 How does the basic scan-line method determine which surfaces are hidden?

SOLUTION

The basic scan-line method looks one at a time at each of the horizontal lines of pixels in the display area. For example, at the horizontal pixel line $y = \alpha$, the graphics data structure (consisting of all scan-converted polygons) is searched to find all polygons with any horizontal (y) pixel values equal to α.

Next, the algorithm looks at each individual pixel in the α row. At pixel (α, β), the depth values (z values) of each polygon found above are compared to find the polygon having the smallest z value at this pixel. The color of pixel (α, β) is then set to the color of the corresponding polygon at this pixel.

10.11 Using the four pixel display and the graphics objects A and B from Prob. 10.8, show how the basic scan-line method would display these objects.

SOLUTION

First we initialize the display to the color y of the back clipping plane (located at $z = 1$).

Frame buffer =

1	y	y
0	y	y
	0	1

1. Set $y = 0$. The scan-converted representations of A and B contain pixels on the $y = 0$ scan line.

   (*a*) Set $x = 0$. Comparing the z values of A and B at pixel (0, 0), we find that the smaller z value is 0, which belongs to A. Thus A is seen from pixel (0, 0); that is, pixel (0, 0) is set to the color of A.

1	y	y
0	a	y
	0	1

   (*b*) Set $x = 1$. Since A is not seen from pixel (1, 0) while B is seen, the color of pixel (1, 0) is set to that of B.

1	y	y
0	a	b
	0	1

2. Set $y = 1$. The scan-converted representations of A and B contain pixels on the $y = 1$ scan line.

   (*a*) Set $x = 0$. Because A is not seen at pixel (0, 1) while B is seen, pixel (0, 1) is set to the color of B.

1	b	y
0	a	b
	0	1

   (*b*) Set $x = 1$. Both A and B are "seen" from pixel (1, 1). The depth of A at pixel (1, 1) is 0, that of B is $\frac{1}{2}$. Thus A is visible at pixel (1, 1).

1	b	a
0	a	b
	0	1

This represents the final image displayed.

10.12 How does edge coherence help to reduce computational effort?

SOLUTION

It is based on the assumption that, if an edge or line intersects a given scan line, it will most likely intersect those scan lines next to it. Thus if the pixels that intersect the edge are to be found, instead of intersecting each scan line with the edge, it is necessary to locate only one intersection pixel and then find the others using the slope m of the edge (see Prob. 10.13).

10.13 Show how the calculation of the intersection of an edge with a scan line can be made incremental as opposed to absolute.

SOLUTION

The absolute calculation requires that we find the x intersection value of the edge (e.g., with equation $y = mx + b$) with the scan line $y = \alpha$ for each α.

The incremental solution to the problem is based on the following observations. Suppose that x_α is the x intersection of the edge with the scan line α. Then the intersection $x_{\alpha+1}$ of the edge with the next scan line $y = \alpha + 1$ can be found as illustrated in Fig. 10-21. From Fig. 10-21, where m is the slope of the edge,

$$\frac{\Delta y}{\Delta x} = \frac{(\alpha + 1) - \alpha}{x_{\alpha+1} - x_\alpha} = m$$

Solving for $x_{\alpha+1}$, we obtain

$$x_{\alpha+1} = x_\alpha + \frac{1}{m}$$

Thus the calculation of the next intersection point is incremental; that is, it is found from the previous intersection point by adding $1/m$ to it.

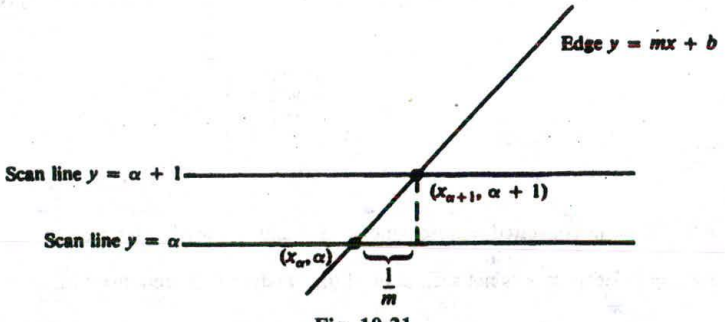


Fig. 10-21

10.14 How does area coherence reduce computational effort?

SOLUTION

Area coherence is based on the assumption that a small enough region of pixels will most likely lie within a single polygon. This reduces, as in the subdivision algorithm, computational effort in dealing with all those polygons that are potentially visible in a given screen area (region of pixels).

10.15 How is spatial coherence determined?

SOLUTION

Spatial coherence is determined by examining the extent of an object. The rectangular extent (bounding box) of an object is determined by finding the minimum and maximum x, y, and z coordinate values of the

points that belong to the object. The extent is the rectangular region bounded by the planes $z = z_{min}$, $z = z_{max}$, $y = y_{min}$, $y = y_{max}$, $x = x_{min}$, and $x = x_{max}$.

10.16 Draw two polygons such that their extents intersect but the polygons themselves don't intersect.

SOLUTION

Figure 10-22 shows one possible solution.

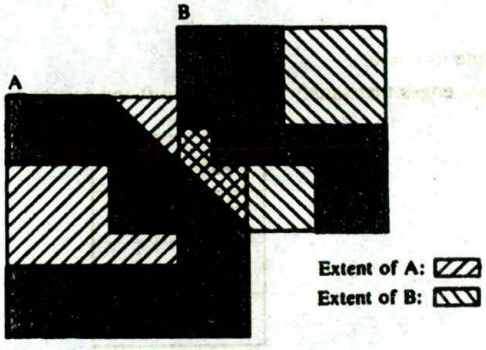


Fig. 10-22

10.17 Apply the scan-line algorithm from Sec. 10.5, under "A Scan-line algorithm," to the display of objects A and B in Prob. 10.8.

SOLUTION

As in Prob. 10.8, a point (x, y) scan converts to the pixel [Floor(x), Floor(y)]. Also note that a nonhorizontal line can be treated as a special polygon: a scanline enters into and exits from the polygon at the same pixel position.

We first construct the edge list (EL) and the polygon list (PL). From Fig. 10-20, the edge entries for the nonhorizontal edges $\overline{A_1A_2}$, $\overline{B_1B_4}$, and $\overline{B_2B_3}$ are

Edge	y_{min}	x	y_{max}	$\frac{1}{m}$	Polygon pointer
$\overline{A_1A_2}$	$\frac{1}{2}$	$\frac{1}{2}$	2	1	A
$\overline{B_1B_4}$	0	0	2	0	B
$\overline{B_2B_3}$	0	2	2	0	B

Here, y_{min} is the edge's smaller y coordinate and x the corresponding x coordinate; y_{max} is the larger y coordinate, and m is the slope of the edge. Form the PL:

PL =

Polygon	Equation	IN/OUT flag	Color
A (line)	$x = y$ $z = 0$	OUT	a
B	$z = \frac{1}{2}$	OUT	b

The algorithm proceeds as follows.

I. *Initialization.*

(*a*) We initialize the screen display to the color y of the back clipping plane (located at $z = 1$).

Frame buffer =

1	y	y
0	y	y
	0	1

(*b*) Set the scan line to $y = 0$.

II. *y-scan loop.* Activate edges that satisfy $\text{Floor}(y_{\min}) = 0$ and sort on x

Edges	x
$\overline{B_1B_4}$	0
$\overline{A_1A_2}$	$\frac{1}{2}$
$\overline{B_2B_3}$	2

III. *x-scan loop.*

(*a*) Process edge $\overline{B_1B_4}$. First invert the appropriate IN/OUT flags in the PL:

PL =

Polygon	Equation	IN/OUT flag	Color
A	$x = y$ $z = 0$	OUT	a
B	$z = \frac{1}{2}$	IN	b

The number of active polygons is one. Thus B is visible at pixel (0, 0) and all pixels on scan line $y = 0$ between edges $\overline{B_1B_4}$ and $\overline{A_1A_2}$ are set to the color of B. In our case, this is just the pixel (0, 0).

1	y	y
0	b	y
	0	1

(*a*) Now we repeat step (*a*), processing edge $\overline{A_1A_2}$. We update the PL:

PL =

Polygon	Equation	IN/OUT flag	Color
A	$x = y$ $z = 0$	IN	a
B	$z = \frac{1}{2}$	IN	b

The number of active polygons is 2. At pixel (0, 0) the depth value (z value) of polygon A is 0 and of

polygon B is $\frac{1}{2}$. Thus polygon A is visible at pixel (0, 0), so the color is set to a.

1	y	y
0	a	y
	0	1

Since A is a line, set the IN/OUT flag of line A to OUT. The color of all pixels between edges $\overline{A_1A_2}$ and $\overline{B_2B_3}$ are determined by the remaining active polygon B.

PL =

Polygon	...	IN/OUT flag	...
A	—	OUT	—
B	—	IN	—

This means that we set the color of pixel (1, 0) to b.

1	y	y
0	a	b
	0	1

(*a*) Again, we repeat step (*a*), processing edge $\overline{B_2B_3}$. We update the PL:

PL =

Polygon	...	IN/OUT flag	...
A	—	OUT	—
B	—	OUT	—

The number of active polygons is 0.

(*b*) Having processed $\overline{B_2B_3}$, the last active edge, we proceed:

(1) All the y_{max} values are equal to 2. These values scan-convert (see Prob. 10.8) to 1. The present y scan line value is 0. So no edges are removed.

(2) Incrementing x by $1/m$,

$\overline{B_1B_4}$	0
$\overline{A_1A_2}$	$\frac{3}{2}$
$\overline{B_2B_3}$	2

(3) Set $y = 1$.

Now we repeat steps II and III.

II. *y-scan loop.* With $y = 1$, no additional edge is activated. The result of sorting active edges remains as above.

III. *x-scan loop.*

(*a*) Process edge $\overline{B_1B_4}$. We update the PL:

PL =

Polygon	...	IN/OUT flag	...
A	—	OUT	—
B	—	IN	—

The number of active polygons is 1. Thus *B* is visible at pixel (0, 1) and all pixels between edge $\overline{B_1B_4}$ and $\overline{A_1A_2}$ on scan line $y = 1$ are set to the color of *B*.

1	*b*	*b*
0	*a*	*b*
	0	1

(*a*) Now we repeat step (*a*), processing edge $\overline{A_1A_2}$. We update the PL:

PL =

Polygon	...	IN/OUT flag	...
A	—	IN	—
B	—	IN	—

There are two active polygons. A depth comparison at pixel (1, 1) shows that it is set to the color of line $\overline{A_1A_2}$.

1	*b*	*a*
0	*a*	*b*
	0	1

Since *A* is a line, we set the IN/OUT flag of *A* to OUT. And we proceed immediately to the next edge at the same pixel location.

(*a*) Again we repeat step (*a*), processing edge $\overline{B_2B_3}$. We update the PL:

PL =

Polygon	Equation	IN/OUT flag	Color
A	$x = y$ $z = 0$	OUT	*a*
B	$z = \frac{1}{2}$	OUT	*b*

The number of active polygons becomes 0.

(*b*) Having processed the last active edge $\overline{B_2B_3}$, we remove those edges for which y_{max} equals the *y* scan value of 1. Since this includes all the edges, i.e., $y_{max} = 2$, the algorithm stops.

10.18 What is the underlying concept of the painter's or the priority algorithm?

SOLUTION

The painter's algorithm sorts polygons by depth and then paints (scan-converts) each polygon onto the screen starting with the most distance polygon.

10.19 What difficulties are encountered in implementing the painter's algorithm?

SOLUTION

First, there is the question of what the "depth" of a polygon is, especially if the polygon is tilted out of the xy plane. Second, if two polygons have the same depth, which one should be painted first?

10.20 If polygon Q has the same depth value as polygon P, which polygon has priority, that is, which is painted first?

SOLUTION

We perform tests 0, 1, 2, 3, 4, 5 (from Sec. 10.4, under "Testing Whether P obscures Q") in order. If any one of the tests is true, we say that polygon P does not obscure polygon Q, and so polygon P is painted first.

If none of the tests is true, we have an ambiguity that must be resolved.

We resolve the ambiguity by switching the roles of P and Q and the reapply tests 0, 1, 2, 3, 4, and 5. If any one of these tests is true, Q does not obscure polygon P and so polygon Q is painted first.

If again none of the tests is true, polygon Q must be subdivided into two polygons Q_1 and Q_2, using the plane containing polygon P as the dividing plane [Fig. 10-4(b)].

10.21 Apply the painter's algorithm to display objects A and B in Prob. 10.8.

SOLUTION

We first find the depth values z_{max} for A and B. Since z_{max} is the largest z value from all the polygon's vertices, then for A, $z_{A_{max}} = 0$, and for B, $z_{B_{max}} = \frac{1}{2}$. Then sorting on z_{max}, we see that polygon B has a higher depth value than polygon A.

Next, we assign priorities by applying tests 0 through 5 in order (see Sec. 10.4). In test 0, the z extent of B is $z_{min} = \frac{1}{2}$, $z_{max} = \frac{1}{2}$. The z extent of A is $z_{min} = 0$, $z_{max} = 0$.

Thus the z extents of A and B do not overlap and $z_{A_{max}}$ is smaller than $z_{B_{min}}$. Thus test 0 is true, and so we scan-convert polygon B first:

Frame buffer =

1	b	b
0	b	b
	0	1

Next, we scan-convert polygon A (i.e., we "paint" over polygon B):

1	b	a
0	a	b
	0	1

This is the final image displayed in the frame buffer.

10.22 What are the basic concepts underlying the subdivision algorithm?

SOLUTION

First, is that a polygon is seen from within a given area of the display screen if the projection of that polygon overlaps the given area. Second, of all polygons that overlap a given screen area, the one that is visible in this area is the one in front of all the others. Third, if we cannot decide which polygon is visible (in

front of the others) from a given region, we subdivide the region into smaller regions until visbility decisions can be made (even if we must subdivide down to the pixel level).

10.23 Apply the subdivision algorithm to the display of objects A and B from Prob. 10.8.

SOLUTION

1. *Initialization.* We initialize the area to the whole screen.
2. *Forming the potentially visible polygon list.* We create the PVPL, sorted on z_{min}, the smallest of the z values of the polygon in the area.

PVPL =

Polygon	z_{min}	Category
A	0	Contained
B	$\frac{1}{2}$	Surrounding

3. *Visibility decision.* We apply criteria (a) through (d) in Sec. 10.6, step 3. Since all four tests fail, we pass to step 4 and subdivide the area into four subregions.

After subdivision, the four newly formed regions are, in our example, the individual pixels. We apply the algorithm to each pixel in turn.

Region 1: pixel (0, 0).

2. *Forming the PVPL.*

PVPL =

Polygon	z_{min}	Category
A	0	Intersecting
B	$\frac{1}{2}$	Surrounding

3. *Visibility decision.* Applying tests (a) thrpough (c), we now apply test (d) since the region is pixel size. The z coordinate of A at pixel (0, 0) is 0, and that of B is $\frac{1}{2}$. Thus A is visible at pixel (0, 0):

Frame buffer =

1		
0	a	
	0	1

Region 2: pixel (0, 1).

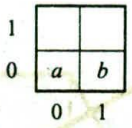
2. *Forming the PVPL.* Note that A is disjoint from this region:

PVPL =

Polygon	z_{min}	Category
B	$\frac{1}{2}$	Surrounding

3. *Visibility decision.* From test (c), there is only one polygon and it is surrounding, so we color pixel (0, 1)

that of B:



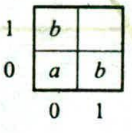
Region 3: pixel (0, 1).

2. *Forming the PVPL.* Since A is disjoint from this region, we have

PVPL =

Polygon	z_{min}	Category
B	$\frac{1}{2}$	Surrounding

3. *Visibility decision.* From test (c), there is only one polygon B and it is surrounding. So region 3 is colored b:



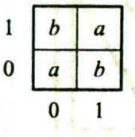
Region 4: pixel (1, 1).

2. *Forming the PVPL.*

PVPL =

Polygon	z_{min}	Category
A	0	Intersecting
B	$\frac{1}{2}$	Surrounding

3. *Visibility decision.* Having applied tests (a) through (c), we now apply test (d). The z coordinate of A at pixel (1, 1) is less than that of B. Thus pixel (1, 1) is set to the color of A:



This is the final image in the frame buffer.

10.24 How can we use the special structure of a convex polyhedron to identify its hidden faces for a general parallel or perspective projection?

SOLUTION

Suppose that on each face of the polyhedron there is an outward-pointing normal vector **N**, attached at a point P of the face (Fig. 10-23). For each face of the polyhedron, let the *line-of-sight vector* **L** be the vector pointing from the face to the viewer. For a parallel projection, this is the direction of projection from the object to the projection plane. For a perspective projection, it is the vector $\overline{\mathbf{PC}}$ from the normal vector attached at point P to the viewpoint at point C (Fig. 10-23).

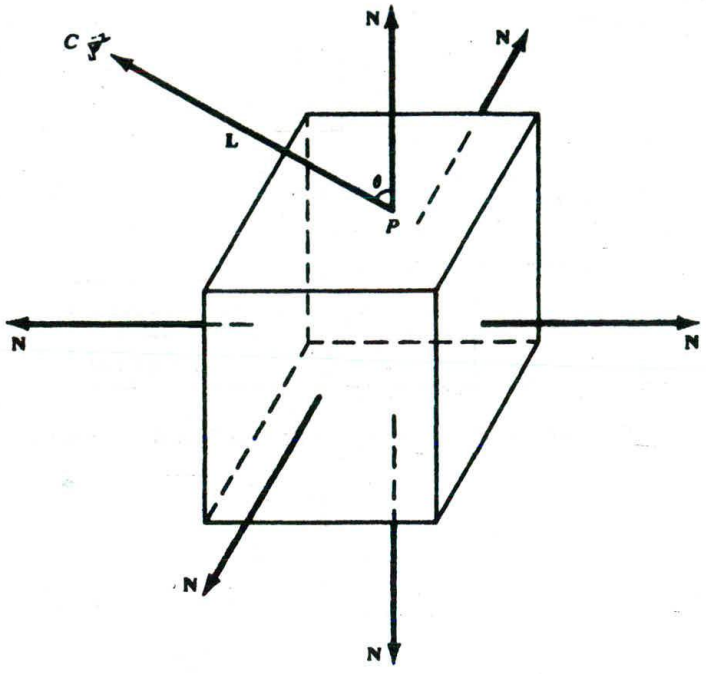


Fig. 10-23

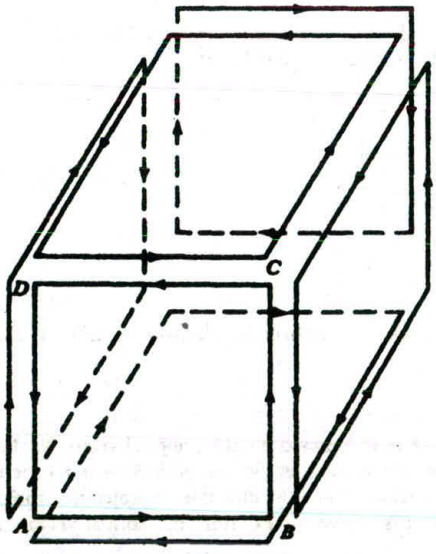


Fig. 10-24

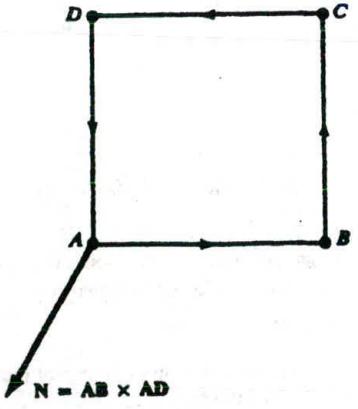


Fig. 10-25

A face is *visible* if the angle θ made by the line-of-sight vector **L** and the normal vector **N** is less than 90°. It is *hidden* if this angle is larger than or equal to 90°. If $0° \leq \theta < 90°$, then $0 < \cos\theta \leq 1$; if $90° \leq \theta \leq 180°$, then $-1 \leq \cos\theta \leq 0$. Since (from App. 2)

$$\cos\theta = \frac{\mathbf{L} \cdot \mathbf{N}}{|\mathbf{L}||\mathbf{N}|}$$

the face is visible if

$$0 < \frac{\mathbf{L} \cdot \mathbf{N}}{|\mathbf{L}||\mathbf{N}|} \leq 1$$

and hidden otherwise. To use this visibility test, we need to find the outward-pointing normal vectors for the faces of the polyhedron. To do this, we label the faces of the polyhedron so that the polyhedron is *oriented*. That is, any edge shared by adjacent faces is traversed in opposite directions with respect to the faces (Fig. 10-24). This guarantees that the normal vectors will point outward. To construct the outward normal vectors, we label each polygon with a counterclockwise labeling (Fig. 10-25). Then a normal vector can be found by taking the cross product of vectors determined by two adjacent sides of the polygon and attaching it at one of the vertices (Fig. 10-25).

Supplementary Problems

10.25 How may the properties of parallel projection be used to simplify hidden-surface calculations for any form of projection?

10.26 What happens when two polygons have the same z value and the Z-buffer algorithm is used?

10.27 How would the Z-buffer algorithm be altered to allow figures to be superimposed on a surface (see Prob. 10.26)?

10.28 Assuming that one allows 2^{24} depth value levels to be used, how much memory would a 1024×768 pixel display require to store the Z buffer?

10.29 How can the amount of computation required by the scan-line method be reduced?

10.30 How does scan-line coherence help to reduce computation?

10.31 What is the extent of the polygon whose vertices are $A(0, 0, 1)$, $B(2, 0, 1)$, and $C(1, 2, 2)$ (see Fig. 10-26)?

10.32 Why are only nonhorizontal lines stored in the edge list of the scan-line algorithm?

10.33 How is the depth of a polygon determined by the painter's algorithm?

10.34 How does the subdivision algorithm exploit are coherence?

10.35 How can hidden-surface algorithms be used to eliminate hidden lines as applied to polygonal mesh models (Chap. 9)?

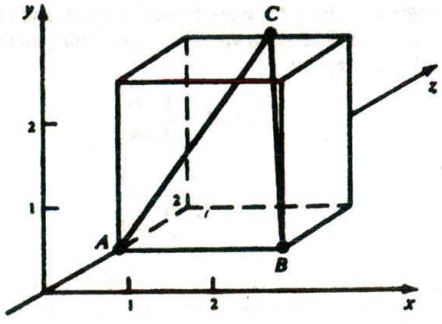


Fig. 10-26

CHAPTER 11

Color and Shading Models

Color is a fundamental attribute of our viewing experience. The perception of color arises from light energy entering our visual system and triggering a chain of not-yet-fully-understood neurological and psychological responses. This complex process is relevant to computer graphics because a realistic image is one that seems indistinguishable from a true record of the light energy coming from the real scene. It has been a constant challenge to find effective and efficient computational models for constructing such images.

In Sect. 11.1, we first outline the basic relationship between light (the physical stimulus) and the perception of color. This leads to an international standard for the specification and measurement of colors, and a better understanding of the widely used RGB color model. In Sect. 11.2, we use a well-known formula to mimic the effect of objects being lit by light sources (A more elaborate illumination model is discussed in Chap. 12). In Sects. 11.3 and 11.4, we describe several useful techniques for shading polygon-mesh objects and for depicting delicate surface details (surface texture).

11.1 LIGHT AND COLOR

Light, or visible light, is electromagnetic energy in the 400 to 700 nm, i.e., nanometer (10^{-9} meter), wavelength (λ) range of the spectrum (see Fig. 11-1). A typical light has its energy distributed across the visible band and the proportions are described by a spectral energy distribution function $P(\lambda)$ (see Fig. 11-2). To model a light or reproduce a given light with physical precision one would need to duplicate the exact energy distribution, which is commonly referred to as spectral reproduction.

On the other hand, it has been shown through carefully designed psychophysical experiments that discrepancy in spectral distribution does not necessarily lead to difference in perception. Less stringent approaches exist to reproduce light to the extent that the reproduction causes the same color sensation to an average human observer as the original. These results make it possible to specify or describe light in ways that are more perceptually oriented and are easier to handle.

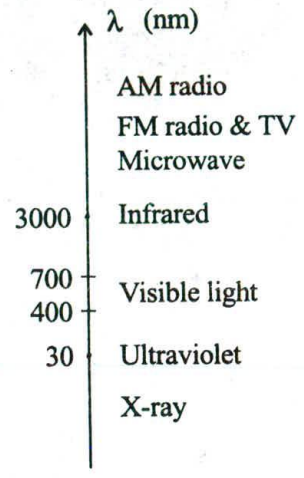


Fig. 11-1

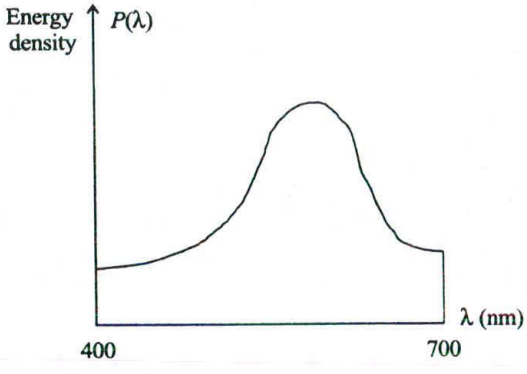


Fig. 11-2

Basic Characteristics of Light

Light can be characterized in three perceptual terms. The first one is *brightness*, which corresponds to its physical property called *luminance*. Luminance measures the total energy in the light. It is proportional to the area bounded by $P(\lambda)$ and the λ axis in the 400 to 700 nm range. The area can be calculated by

$$\int_{\lambda} P(\lambda)\mathrm{d}\lambda$$

The higher the luminance, the brighter the light to the observer.

The second perceptual term is *hue*, which distinguishes a white light from a red light or a green light. For a light with an idealized spectral distribution as shown in Fig. 11-3, hue corresponds to another physical property called the *dominant wavelength* of the distribution.

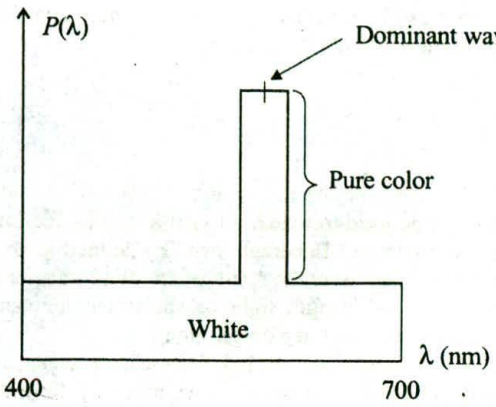


Fig. 11-3

The third perceptual term is *saturation*, which describes the degree of vividness. For example, two red lights may differ in luminance/brightness, and they may differ in degree of vividness (e.g., pure/saturated red vs. pale/unsaturated red). Saturation corresponds to the physical property called *excitation purity*, which is defined to be the percentage of luminance that is allocated to the dominant or pure color component (see Fig. 11-3). In other words, we have

$$\text{saturation} = \frac{\text{pure color}}{\text{pure color} + \text{white color}}$$

Although this simple scheme has its weaknesses (e.g., not all lights have an identifiable dominant wavelength), it bridges the physical aspects of light and the perceptual classification of color in a straightforward fashion.

The Trichromatic Generalization Theory

Let S be a given light or color stimulus. The effect of S (the color sensation of a human subject observing S) can be matched by combining light from three primary sources in appropriate proportions:

$$S = r \cdot \text{red} + g \cdot \text{green} + b \cdot \text{blue}$$

In other words, the given light and the proportional mix of the three primaries look the same to the observer.

Note that the theory stands with any triple of primaries. Three light sources form a triple of primaries as long as none of the three can be expressed as/matched by a linear combination of the other two. We use red, green, and blue in the formula mainly because they are the standard choice in color-matching experiments (with red = 700 nm, green = 546.1 nm, and blue = 435.8 nm). Moreover, these three colors roughly coincide with the wavelength values that cause peak response from the three types of color-sensitive receptor cells in the retina, a membrane that lines the back of the eye's wall. These receptor cells are called β, δ, and ρ cones, respectively. They are most sensitive to light in the wavelength range of β: 440–445 nm, δ: 535–545 nm, and ρ: 575–580 nm. Another kind of receptor cells called rods are color-blind but are very sensitive to low intensity light.

A critical aspect of this tri-stimulus approach is that, in order to match all visible colors, the weight values sometimes have to be negative. For example, when red, green, and blue are used as primaries the value of b, the weight of the blue component, may be negative. A negative b value means that the effect of the given stimulus S cannot be matched normally through the additive process. But if S is mixed with some

blue light, then the effect of the mix can be matched by a linear combination of red and green (Intuitively, we move the term $b \cdot$ blue to the other side of the equation).

CIE *XYZ* Color Model

Looking for a good compromise between the simple and effective tri-stimulus method and the fact that no naturally occurring primaries can be used to match all visible colors, the International Commission on Illumination (Commission Internationale de l'Eclairage, or CIE) defined three imaginary (non-realizable) primaries, vis., $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$, in 1931. They were the result of an affine transformation applied to three real primaries to ensure that every single-wavelength light or spectral color can be expressed as a linear combination of the CIE primaries without any negative weight.

The relative amounts of $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ that are needed to describe each spectral color are shown in Fig. 11-4 in the form of three color-matching functions $x(\lambda)$, $y(\lambda)$, and $z(\lambda)$. In order to match a color light of wavelength λ_0, the proper proportion is found by the line $\lambda = \lambda_0$ intersecting the curves representing the three functions. In addition, $y(\lambda)$ matches the *luminous efficiency* of the human eye and corresponds to the eye's response to light of constant luminance.

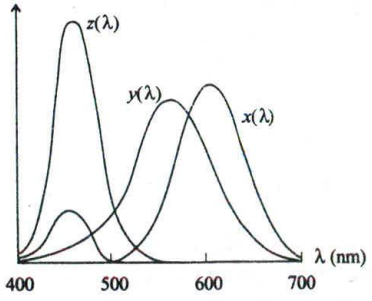


Fig. 11-4

To describe an arbitrary light S with spectral distribution $P(\lambda)$, we "add together" the respective amounts of the CIE primaries that are necessary to match all the spectral components of S. This is done with

$$X = k\int_{\lambda} P(\lambda)x(\lambda)\,d\lambda, \qquad Y = k\int_{\lambda} P(\lambda)y(\lambda)\,d\lambda, \qquad Z = k\int_{\lambda} P(\lambda)z(\lambda)d\lambda$$

where k is a light-source-dependent constant. The resultant X, Y (which carries luminance information due to the way $y(\lambda)$ is defined), and Z are used as weights to express S as follows:

$$S = X \cdot \boldsymbol{X} + Y \cdot \boldsymbol{Y} + Z \cdot \boldsymbol{Z}$$

CIE Chromaticity Diagram

Now we define x, y, and z by normalizing the above weights against $X + Y + Z$:

$$x = \frac{X}{X+Y+Z}, \qquad y = \frac{Y}{X+Y+Z}, \qquad z = \frac{Z}{X+Y+Z}$$

Clearly $x + y + z = 1$, and we have $z = 1 - x - y$. The two variables x and y represent colors by grouping them into sets, each of which has members that differ only in luminance.

Using x and y as the horizontal and vertical axes we can plot the well-known CIE Chromaticity Diagram (see Fig. 11-5). The curved triangular figure encompasses all perceivable colors by ignoring luminance. Only when two colors differ in chromaticity (hue and/or saturation) are they represented by two different points in the diagram (i.e., they have different chromaticity coordinates).

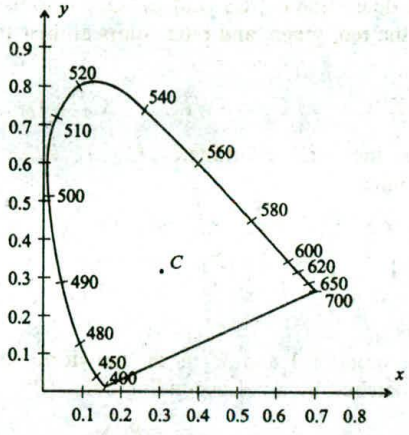


Fig. 11-5

To convert from chromaticity coordinates (x, y) back to a specific color in the XYZ color space we need an additional piece of information, typically Y:

$$X = \frac{x}{y}Y, \qquad Y = Y, \qquad Z = \frac{1-x-y}{y}Y$$

We can find all spectral colors along the figure's upper border. Point C $(0.310, 0.316)$ is known as *Illuminant C*. It corresponds to a reference white that is obtained from a spectral distribution close to daylight.

The CIE chromaticity diagram serves as a universal reference for the specification and comparison of visible colors. The phosphors of a typical color monitor have the following chromaticity coordinates: R $(0.62, 0.34)$, G $(0.29, 0.59)$, and B $(0.15, 0.06)$. These coordinates define a triangular region within the diagram. The region is referred to as the *color gamut* of the monitor, which represents (the chromaticity range of) all the colors this monitor is able to display.

Color Gamut Mapping

Let (x_r, y_r), (x_g, y_g), and (x_b, y_b) be the chromaticity coordinates of the phosphors of an RGB color monitor. What are the XYZ coordinates of its colors? We first introduce auxiliary variables $C_r = X_r + Y_r + Z_r$, $C_g = X_g + Y_g + Z_g$, and $C_b = X_b + Y_b + Z_b$, where (X_r, Y_r, Z_r), (X_g, Y_g, Z_g), and (X_b, Y_b, Z_b) denote the respective XYZ coordinates of the red, green, and blue colors the monitor can display. We have

$$\begin{aligned}
X_r &= x_r C_r, & Y_r &= y_r C_r, & Z_r &= z_r C_r & \text{where } z_r &= 1 - x_r - y_r \\
X_g &= x_g C_g, & Y_g &= y_g C_g, & Z_g &= z_g C_g & \text{where } z_g &= 1 - x_g - y_g \\
X_b &= x_b C_b, & Y_b &= y_b C_b, & Z_b &= z_b C_b & \text{where } z_b &= 1 - x_b - y_b
\end{aligned}$$

Thus the *XYZ* coordinates of a composite *RGB* color can be expressed in terms of a transformation M:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = M \begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} x_r C_r & x_g C_g & x_b C_b \\ y_r C_r & y_g C_g & y_b C_b \\ z_r C_r & z_g C_g & z_b C_b \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} x_r & x_g & x_b \\ y_r & y_g & y_b \\ z_r & z_g & z_b \end{pmatrix} \begin{pmatrix} C_r & 0 & 0 \\ 0 & C_g & 0 \\ 0 & 0 & C_b \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

There are two different ways to determine C_r, C_g, and C_b. One is to use a photometer to measure the luminance levels Y_r, Y_g, and Y_b of the red, green, and blue colors at their maximum intensity/brightness, respectively. Then

$$C_r = Y_r/y_r, \qquad C_g = Y_g/y_g, \qquad C_b = Y_b/y_b$$

The other is to find/measure the *XYZ* coordinates (X_w, Y_w, Z_w) of the monitor's white color $(R = G = B = 1)$. Using these we have

$$\begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} = \begin{pmatrix} x_r & x_g & x_b \\ y_r & y_g & y_b \\ z_r & z_g & z_b \end{pmatrix} \begin{pmatrix} C_r \\ C_g \\ C_b \end{pmatrix}$$

We can now solve for C_r, C_g, and C_b.

If M_1 is the transformation for monitor 1 and M_2 is the transformation for monitor 2, then a color (R_1, G_1, B_1) on monitor 1 can be matched by a corresponding color (R_2, G_2, B_2) on monitor 2:

$$\begin{pmatrix} R_2 \\ G_2 \\ B_2 \end{pmatrix} = M_2^{-1} M_1 \begin{pmatrix} R_1 \\ G_1 \\ B_1 \end{pmatrix}$$

provided that (R_2, G_2, B_2) is within the color gamut of monitor 2.

The NTSC *YIQ* Color Model

The NTSC (National Television System Committee) *YIQ* color model is used for commercial television broadcasting in the US. It is defined to be a linear transformation of the *RGB* color model, with the Y component purposely made to be the same as the Y in the CIE *XYZ* color model. Since Y carries luminance information, black-and-white television sets can use it to display color images as gray-scale pictures. The mapping between *RGB* and *YIQ* is as follows:

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

The quantities in the transformation matrix are obtained using the standard NTSC *RGB* phosphors whose chromaticity coordinates are R (0.67, 0.33), G (0.21, 0.71), and B (0.14, 0.08). It is also assumed that the white point is at $x_w = 0.31$, $y_w = 0.316$, and $Y_w = 1.0$.

11.2 THE PHONG MODEL

This is a widely used and highly effective way to mimic the reflection of light from object surfaces to the viewer's eye. It is considered an empirical approach because, although it is consistent with some basic principles of physics, it is largely based on our observation of the phenomenon. It is also referred to as a *local illumination model* because its main focus is on the direct impact of the light coming from the light source. On the other hand, a *global illumination model* attempts to include such secondary effects as light going through transparent/translucent material and light bouncing from one object surface to another.

Now consider a point light source [see Fig. 11-6(*a*)], which is an idealized light with all its energy coming out from a single point in space (a reasonable approximation to a bulb). Our eye is at the viewpoint

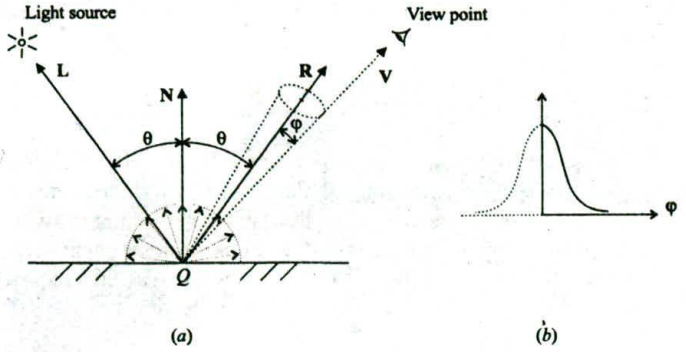


Fig. 11-6

looking at point Q on the surface of an object. What should be the color of Q? In other words, what should be the color of the light reflected into our eye from Q (in the direction of vector **V**)?

There are two extreme cases of light reflection. The first is called *diffuse reflection*. In this case light energy from the light source (in the direction of $-\mathbf{L}$) gets reflected/bounced off equally in all directions (see the small arrows forming a half-circle/hemisphere). We also want the energy level of the reflection to be a function of the incident angle θ (between **L** and surface normal **N**). The smaller the angle, the higher the reflection (kind of like bouncing a ball off a wall). The mathematical tool we use to achieve these is to have the reflection proportional to $\cos(\theta)$.

The second case is called *specular reflection*. It attempts to capture the characteristic of a shiny or mirror-like surface. Were the surface in Fig. 11-6(*a*) a perfect mirror, energy from the light source would be reflected in exactly one direction (the direction of vector **R**). Since a perfect mirror is nonexistent we want to distribute reflected energy across a small cone-shaped space centered around **R**, with the reflection being the strongest along the direction of **R** (i.e., $\varphi = 0$) and decreasing quickly as φ increases [see the bell-shaped curve in Fig. 11-6(*b*)]. The mathematical means for modeling this effect is $\cos^k(\varphi)$, where the parameter k provides for a convenient way to vary the degree of shininess ($k = 1$ for a dull surface and $k = 100$ for a mirror-like surface). For a given scene we can find out the amount of specular reflection in the direction of **V** using the actual angle φ between **R** and **V**.

Furthermore, the complex inter-object reflection needs to be accounted for in some way because many surfaces we see are not lit directly by the light source. They are lit by light that is bouncing around the environment. For this we introduce a directionless ambient light, which illuminates all surfaces in the scene and gets reflected uniformly in all directions by each surface.

Thus in the Phong model, object surfaces are thought to produce a combination of ambient-light reflection and light-source-dependent diffuse/specular reflection. Mathematically we can state the total reflected energy intensity I as

$$I = I_a k_a + I_p(k_d \cos(\theta) + k_s \cos^k(\varphi))$$

where I_a is the intensity of the ambient light; I_p is the intensity of the point light; and $0 \leq k_a, k_d, k_s \leq 1.0$ are reflection coefficients that represent the surface's ability to reflect ambient light, to produce diffuse reflection, and to produce specular reflection, respectively (e.g., $k_a = 0.2$ means 20% reflection of I_a).

If **L**, **N**, **R**, and **V** are all unit vectors, then $\mathbf{L} \cdot \mathbf{N} = \cos(\theta)$ and $\mathbf{R} \cdot \mathbf{V} = \cos(\varphi)$. We can write the formula as

$$I = I_a k_a + I_p(k_d \mathbf{L} \cdot \mathbf{N} + k_s(\mathbf{R} \cdot \mathbf{V})^k)$$

Note that the term $\mathbf{L} \cdot \mathbf{N}$ is used in computing $\mathbf{R}$ (see Prob. 11.10). When there are two or more light sources in the scene, their effects are cumulative:

$$I = I_a k_a + \sum_{i=1}^{n} I_{p_i}(k_d \mathbf{L}_i \cdot \mathbf{N} + k_s(\mathbf{R}_i \cdot \mathbf{V})^k)$$

These formulas are typically used along with the *RGB* color model. Thus light intensity is in the form of an *RGB* color vector, e.g., $I = (I_r, I_g, I_b)$. Reflection coefficients are also three-dimensional vectors. For example, $k_d = (0.7, 0.7, 0.3)$ defines a surface that looks yellowish when illuminated with white light. The ambient reflection coefficient k_a can simply be the same as k_d. The three components of k_s are often made equal since the color of the reflection of a light source is typically the same as the color of the light source itself. When the light is white, the formula for a single point source becomes

$$\begin{cases} I_r = I_a k_{ar} + I_p(k_{dr} \mathbf{L} \cdot \mathbf{N} + k_s(\mathbf{R} \cdot \mathbf{V})^k) \\ I_g = I_a k_{ag} + I_p(k_{dg} \mathbf{L} \cdot \mathbf{N} + k_s(\mathbf{R} \cdot \mathbf{V})^k) \\ I_b = I_a k_{ab} + I_p(k_{db} \mathbf{L} \cdot \mathbf{N} + k_s(\mathbf{R} \cdot \mathbf{V})^k) \end{cases}$$

Figure 11-7 shows a gray scale image with 16 views of a sphere shaded using the Phong formula. The four rows from top to bottom are calculated using $k_d/k_s = 0.5/0.1$, $0.5/0.3$, $0.5/0.5$, and $0.5/0.7$, respectively. The four columns from left to right represent $k = 3$, 5, 20, and 100, respectively.

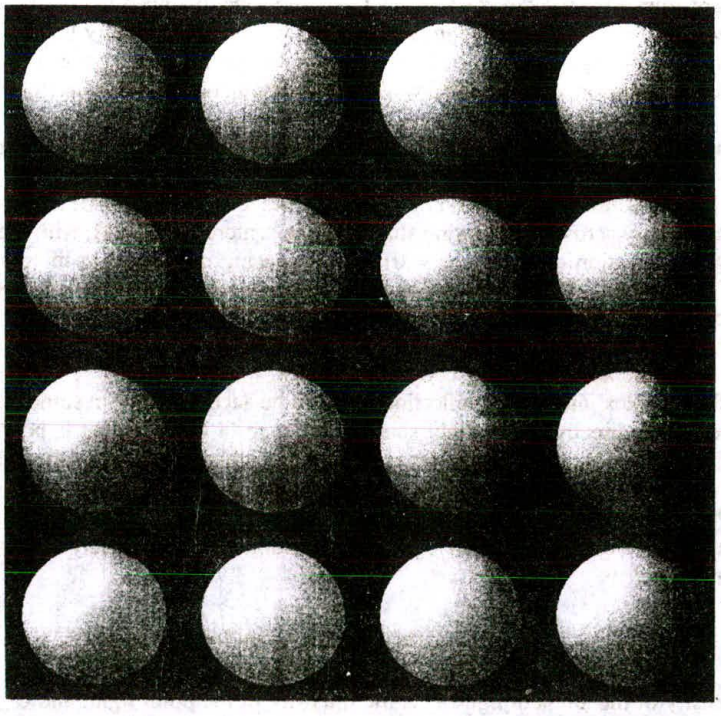


Fig. 11-7

11.3 INTERPOLATIVE SHADING METHODS

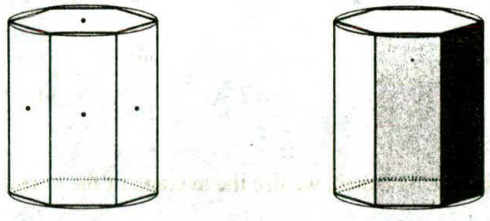
Computing surface color using an illumination model such as the Phong formula at every point of interest can be very expensive. It is even unnecessary when the image is used to preview the scene. To

circumvent this problem, we apply the formula at full scale only at selected surface points. We then rely on such techniques as color interpolation and surface-normal interpolation to shade other surface points.

Constant Shading

The least time-consuming approach is not to perform calculation for additional surface points at all. The color values of those selected surface points are used to shade the entire surface.

For example, a cylindrical object may be modeled by a polygon mesh as shown in Fig. 11-8(*a*). We can evaluate the Phong formula using the geometric center of each polygon and set the resultant color to every pixel that belongs to the corresponding polygon [see Fig. 11-8(*b*)].



(*a*) (*b*)

Fig. 11-8

Constant shading can produce good results for dull polyhedrons lit by light sources that are relatively far away (hence the diffuse component varies little within each polygonal face). The primary weakness of this method is that false contours appear between adjacent polygons that approximate a curved surface [see the cylindrical object in Fig. 11-8(*b*)]. In addition, the specular reflection of the light source (often referred to as the *specular highlight*) tends to get lost. On the other hand, if a selected surface point happens to be at the location of the strongest specular reflection of the light source, then the color of the corresponding polygon will be, for the most part, significantly distorted.

Gouraud Shading

In this approach we evaluate the illumination formula at the vertices of the polygon mesh. We then interpolate the resultant color values to get a gradual shading within each polygon.

For example, we use bilinear interpolation to find the color of point P inside a triangle whose vertices are P_1, P_2, and P_3 (see Fig. 11-9). The scan line containing P intersects the two edges at points P' and P''. We interpolate the color of P_1 and the color of P_2 to get the color of P'. We then interpolate the color of P_2 and the color of P_3 to get the color of P''. Finally, we interpolate the color of P' and the color of P'' to get the color of P (see Prob. 11.15).

When polygons are used to approximate a curved surface, there are two ways to determine the normal vectors at the vertices in order to get a smooth-looking transition between adjacent polygons. The first is to use the underlying curved surface to find the true surface normal at each vertex location. For example, the vertices of the polygon mesh in Fig. 11-8(*a*) are on the surface of an underlying cylinder. The normal vector at a point on the cylinder is perpendicular to the axis of the cylinder and points from the axis to the point [see Fig. 11-10(*a*)].

On the other hand, it may sometimes be difficult to find normal vectors from the underlying surface, or the polygon mesh may not come from any underlying surface. A second approach to deciding normal vectors at the vertices is to average the normal vectors of adjacent polygons. For example, to determine

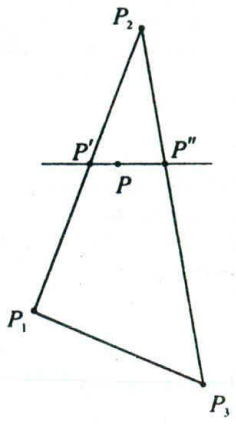


Fig. 11-9

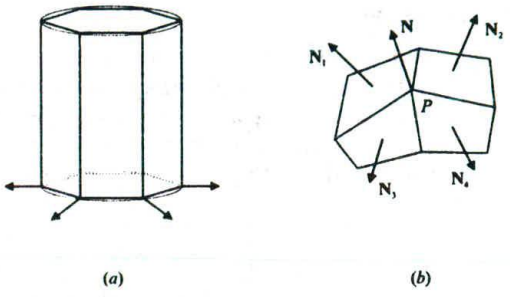


Fig. 11-10

normal vector **N** at vertex **P** in Fig. 11-10(*b*), we use the average of the normal vectors of the polygons that meet at *P*:

$$\mathbf{N} = \mathbf{N}_1 + \mathbf{N}_2 + \mathbf{N}_3 + \mathbf{N}_4$$

where **N** is then normalized by dividing it by $|\mathbf{N}|$.

Common normal vectors at the vertices of a polygon mesh that approximates a curved surface result in color values that are shared by adjacent polygons, eliminating abrupt changes in shading characteristics across polygon edges. Even when the polygon mesh is relatively coarse, Gouraud shading is very effective in suppressing false contours. However, a much finer mesh is needed in order to show reasonably good specular reflection.

Phong Shading

Instead of color values, we may also interpolate normal vectors. For example, to find the normal vector **N** at point *P* in a triangle (see Fig. 11-11), we first interpolate $\mathbf{N}_1$ and $\mathbf{N}_2$ to get $\mathbf{N}'$, then interpolate $\mathbf{N}_2$ and $\mathbf{N}_3$ to get $\mathbf{N}''$, finally interpolate $\mathbf{N}'$ and $\mathbf{N}''$ to get **N**.

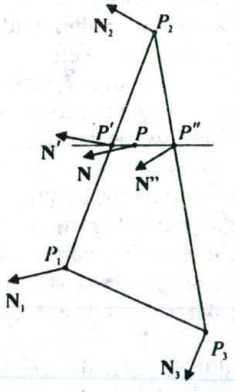


Fig. 11-11

This technique is relatively time-consuming since the illumination model is evaluated at every point of interest using the interpolated normal vectors. However, it is very effective in dealing with specular highlights.

11.4 TEXTURE

While gradual shading based on illumination is an important step towards achieving photo-realism, most real-life objects have regular or irregular surface features (e.g., wood grain). These surface details are collectively referred to as *surface texture*. In this section we present three approaches to adding surface texture: *projected texture*, *texture mapping*, and *solid texture*.

Projected Texture

Imagine putting a plain object between a slide projector and the projection screen. The surfaces of the object that face the projector will be covered by the projected image. If the image depicts wood grain then the affected surfaces will have the intricate wood texture superimposed onto its original shading.

In computer graphics, projecting texture onto an object is effectively the inverse of projecting an object onto the view plane (compare Fig. 7-1 and Fig. 11-12) . We now refer to the view plane as the reference plane, which contains a two-dimensional texture definition called the *texture map*. The projection line that associates point P with its image P' allows the shading of P to be determined or influenced by information in the texture map at location P'.

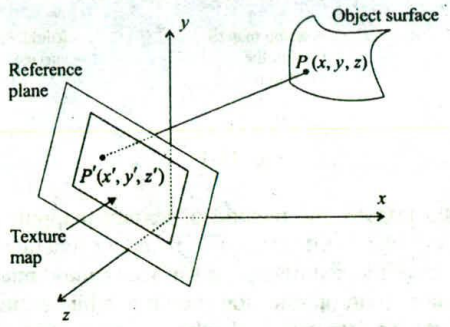


Fig. 11-12

The texture map is often a synthesized or scanned image, in which case it contains color attributes. There are several ways for the color attributes of P' to be used to shade point P. First, we may simply assign these attributes to P, replacing whatever color attributes P currently has. The net effect of this approach is to paint the color of the texture onto the object. Second, we may interpolate between the color C' of P' and the original color C of P using $(1 - k)C + kC'$ where $0 \leq k \leq 1.0$ to come up with a new color for P. The result of this calculation makes the texture appear blended into the original shading. Third, we may use a logical operation involving C and C' to compute a new color for P. For example, when the AND operation is used, a texture area with magenta shades will appear unaltered if the original color is white, but it will take on red shades if the original is yellow.

Projected texture is an effective tool when target surfaces are relatively flat and facing the reference plane. On the other hand, "hidden surfaces" are not affected, and features of the texture are distorted by the curvature of the surface.

Texture Mapping

Now imagine the texture map being a thin and flexible sheet made of elastic material. When we wrap it around the surface of an object we can stretch or shrink it so as to follow the shape of the object.

This approach is referred to as *texture mapping*. We describe the texture map in a two-dimensional space called the *texture space* with coordinates (u, w), and we represent the surface of an object in parametric form using (θ, φ). The mapping from the texture space to the object space is then defined by

$$\theta = f(u, w) \qquad \varphi = g(u, w)$$

and the inverse mapping by

$$u = r(\theta, \varphi) \qquad w = s(\theta, \varphi)$$

Although not necessary, these mapping functions are generally assumed to be linear (so the texture map stretches and shrinks proportionally). Thus we can write

$$\theta = Au + B \qquad \varphi = Cw + D$$

where the constants A, B, C, and D are obtained from the relationship between known points in the two spaces (e.g., corners of the texture map and the corresponding surface points).

As mentioned early, the texture map often contains color attributes that can be used in different ways to alter the original shading. In addition, its role may be extended to provide other kind of information that affects the outcome of the shading process. For example, a technique called *bump mapping* uses the quantities in the texture map to displace or perturb the normal vectors of the surface to produce an appearance of roughness (see Fig. 11-13 for a one-dimensional illustration of the effect).

Smooth-looking surface

A bump map to perturb the magnitude and direction of the surface normals

Rough-looking surface

Fig. 11-13

In comparison with projected texture, the method of texture mapping can better adapt to a curved surface. However, complex shapes deter the derivation of mapping functions. Furthermore, both methods are particularly weak when features of the texture on one surface should "match" those on others. Consider a wood cube, for instance: the wood grain on one side should exhibit continuity and consistency with the wood grain on all adjacent sides. As we turn the wood cube into a sculpture, the complexity of this texture-matching problem simply multiplies.

Solid Texture

The weaknesses of projected texture and texture mapping largely stem from the inherent mismatch between a two-dimensional texture map and a three-dimensional surface. In this approach we define texture using a three-dimensional texture space. The texture definition, referred to as *solid texture*, is a three-dimensional representation of the internal structure of some nonhomogeneous material such as wood, marble, etc.

When rendering an object made of a specific material, we use transformations to place the object into the coordinate system where the texture for the material is defined. In other words, we map a surface point $P(x, y, z)$ to its coordinates (u, v, w) in the texture space. Information associated with this location is then used to control the shading of P. This makes the object look as if it were carved out of a solid piece of the chosen material.

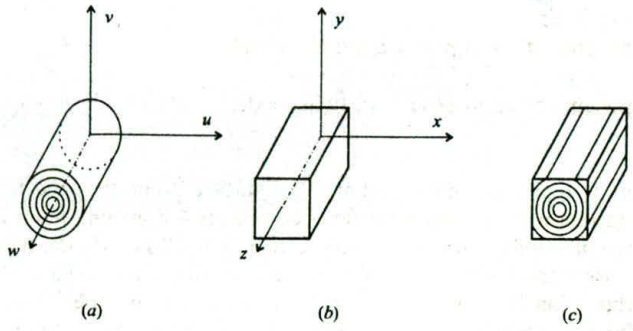


Fig. 11-14

Solid texture is often defined procedurally. The following is an example that shows the development of a model for wood grain. We begin by representing the most prominent feature of the material with a series of concentric cylinders in the texture space [see Fig. 11-14(*a*)]) . When an object such as the parallelepiped in Fig. 11-14(*b*) is placed into the texture field, the object intersects the cylinders, leaving intersection contours on its surfaces. The contours we get by aligning the object coordinate axes with the texture coordinate axes are shown in Fig. 11-14(*c*). We now rotate the front end of the parallelepiped slightly to the left about the *v* axis, and slightly up about the *u* axis. The result is our first approximation to wood grain (see Fig. 11-15).

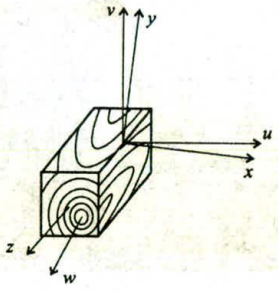


Fig. 11-15

We can see that the contours on different sides are consistent with the underlying three-dimensional structure, but they are just too smooth and too perfect to look real. In order to incorporate some asymmetry and irregularity we need to bring two updates to the model. The first is to add ripples or sinusoidal perturbations to the cylinders so the intersection contours will have some minor and natural-looking oscillations. The second is to turn or twist the ripples slowly along the *w* axis to give the contours a skewed appearance. An implementation of the improved cylinder model is shown in the following pseudo-code procedure:

```
woodGrain(pt, kd)
{
  float radius, angle;
  radius = sqrt(pt[u]^2 + pt[v]^2);
  angle = (pt[u] == 0) ? π/2 : arctan(pt[v]/pt[u]);
  radius = radius + m*sin( f*angle + r*pt[w]);
```

```
    if ((int)radius % p < d)
      set the components of kd to produce a dark shade;
    else
      set the components of kd to produce a light shade;
}
```

where array parameter *pt* represents the coordinates of surface point *P* in the texture space, and array parameter *kd* is used to return the proper reflection coefficients for shading point *P*.

The first instruction in woodGrain is to determine the radius of the cylinder that intersects *P*. The next instruction sets the base parameter value for the sine function that delivers the ripple effects. The magnitude of the perturbation is controlled by *m*, the frequency or number of ripples in a full circle is controlled by *f*, and the rate at which the ripples rotate along the *w* axis is controlled by *r*. Finally, we treat wood grain as alternating bands of dark and light cylinders of period *p*. Within a period the two bands are separated by threshold *d*. Although only two shades are shown in the procedure, the value of *kd* should be made to vary based on the relative position of *P* within each band to produce better results (see Fig. 11-16 for a ray-traced sphere).

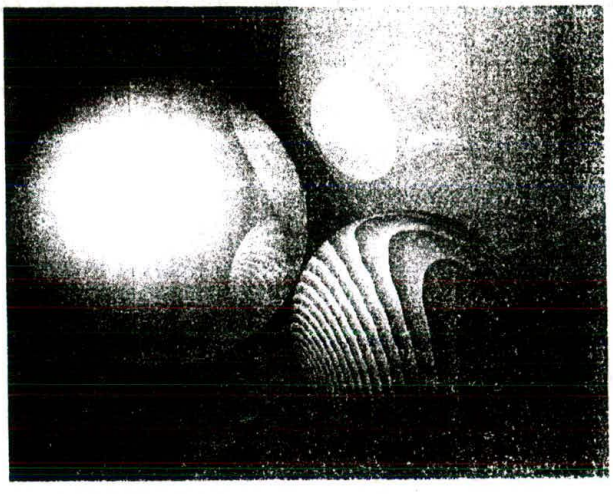


Fig. 11-16

Solved Problems

11.1 Assuming that the medium of interest is air (or a vacuum) express the visible band of the spectrum in terms of a frequency range.

SOLUTION

Since frequency = speed of light/wavelength, and speed of light $\approx 3.0 \times 10^8$ m/s, we obtain the frequency range $(3.0 \times 10^8)/(700 \times 10^{-9})$ to $(3.0 \times 10^8)/(400 \times 10^{-9})$ Hz or 4.3×10^{14} to 7.5×10^{14} Hz.

11.2 Name the three perceptual terms for describing color and the corresponding physical properties of light.

SOLUTION

Brightness/luminance, hue/dominant wavelength, saturation/excitation purity.

11.3 Derive a simple formula to calculate the area bounded by the distribution function $P(\lambda)$ in Fig. 11-3.

SOLUTION

Let W and H be the width and height of the rectangular area that corresponds to white, and D and P be the width and height of the rectangular area that corresponds to pure color. The area bounded by $P(\lambda)$ is $W \times H + D \times P$.

11.4 Name the two kinds of receptor cells in the retina and describe their basic function.

SOLUTION

They are cones for seeing colors, and rods for perceiving low intensity light.

11.5 Why did we say that red, green, and blue only *roughly* coincide with the wavelength values that cause peak response from the three types of color-sensitive cones?

SOLUTION

The three colors have the following typical wavelength: red = 700 nm, green = 546.1 nm, and blue = 435.8 nm. Although blue and green closely match the sensitive range of two types of cones, β : 440–445 nm and δ : 535–545 nm, the sensitive range of the third type, ρ : 575–580 nm, is actually yellow, not red.

11.6 Do we have to use Y in order to convert from chromaticity coordinates (x, y) back to a specific color in the XYZ color space?

SOLUTION

No. If we know X, we can use

$$X = X, \qquad Y = \frac{y}{x}X, \qquad Z = \frac{1-x-y}{x}X$$

11.7 Presume that a monitor produces what is called the standard white D_{65} with $x_w = 0.313$, $y_w = 0.329$, and $Y_w = 1.0$ when $R = G = B = 1$, and the chromaticity coordinates of its phosphors are as given in Sec. 11.1. Find the color transformation matrix M for the monitor.

SOLUTION

Since $Y_w = 1.0$, we have

$$X_w = x_w/y_w = 0.951 \qquad \text{and} \qquad Z_w = (1 - x_w - y_w)/y_w = 1.088$$

Use these and the chromaticity coordinates of the phosphors

$$\begin{pmatrix} 0.951 \\ 1.0 \\ 1.088 \end{pmatrix} = \begin{pmatrix} 0.62 & 0.29 & 0.15 \\ 0.34 & 0.59 & 0.06 \\ 0.04 & 0.12 & 0.79 \end{pmatrix} \begin{pmatrix} C_r \\ C_g \\ C_b \end{pmatrix}$$

Solve and get $C_r = 0.705$, $C_g = 1.170$, and $C_b = 1.164$. Now

$$M = \begin{pmatrix} x_r C_r & x_g C_g & x_b C_b \\ y_r C_r & y_g C_g & y_b C_b \\ z_r C_r & z_g C_g & z_b C_b \end{pmatrix} = \begin{pmatrix} 0.437 & 0.339 & 0.175 \\ 0.240 & 0.690 & 0.070 \\ 0.028 & 0.140 & 0.920 \end{pmatrix}$$

11.8 Verify the fact that the Y in the CIE XYZ color model is the same as the Y in the NTSC YIQ color model.

SOLUTION

Find the transformation to XYZ for the standard NTSC RGB display (see Sec. 11.1). Since $x_w = 0.31, y_w = 0.316$, and $Y_w = 1.0$, we have

$$X_w = x_w/y_w = 0.981 \qquad \text{and} \qquad Z_w = (1 - x_w - y_w)/y_w = 1.184$$

Use these and the chromaticity coordinates of the standard NTSC phosphors

$$\begin{pmatrix} 0.981 \\ 1.0 \\ 1.184 \end{pmatrix} = \begin{pmatrix} 0.67 & 0.21 & 0.14 \\ 0.33 & 0.71 & 0.08 \\ 0.0 & 0.08 & 0.78 \end{pmatrix} \begin{pmatrix} C_r \\ C_g \\ C_b \end{pmatrix}$$

Solve and get $C_r = 0.906$, $C_g = 0.826$, and $C_b = 1.432$. Now

$$M = \begin{pmatrix} x_r C_r & x_g C_g & x_b C_b \\ y_r C_r & y_g C_g & y_b C_b \\ z_r C_r & z_g C_g & z_b C_b \end{pmatrix} = \begin{pmatrix} 0.607 & 0.173 & 0.200 \\ 0.299 & 0.587 & 0.114 \\ 0.0 & 0.066 & 1.117 \end{pmatrix}$$

Since the middle row of M is the same as the top row of the matrix for mapping from RGB to YIQ, we can see that the Y in CIE XYZ is the same as the Y in NTSC YIQ.

11.9 What is the difference between a local illumination model and a global illumination model?

SOLUTION

A local illumination model focuses on the direct impact of the light coming from the light source. On the other hand, a global illumination model attempts to include such secondary effects as light going through transparent/translucent material and light bouncing from one object surface to another.

11.10 Refer to Fig. 11-6 in Sec. 11.2. Find a formula to compute **R**, the reflection of vector **L** with respect to normal vector **N**.

SOLUTION

Introducing auxiliary vectors **e** and **m** (see Fig. 11-17), we can write

$$\mathbf{R} = \mathbf{m} - \mathbf{e}$$

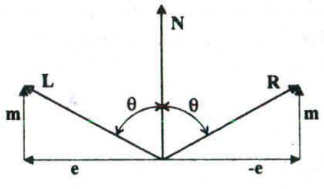


Fig. 11-17

Since $\mathbf{L} = \mathbf{e} + \mathbf{m}$, we have $\mathbf{e} = \mathbf{L} - \mathbf{m}$ and

$$\mathbf{R} = \mathbf{m} - (\mathbf{L} - \mathbf{m}) = 2\mathbf{m} - \mathbf{L}$$

Note that **m** is simply the perpendicular projection of **L** onto **N** (see Prob. A2.16) and **N** is a unit vector, hence

$$\mathbf{m} = (\mathbf{L} \cdot \mathbf{N})\mathbf{N}$$

Finally we have

$$\mathbf{R} = 2(\mathbf{L} \cdot \mathbf{N})\mathbf{N} - \mathbf{L}$$

11.11 The term $\mathbf{R} \cdot \mathbf{V}$ in the Phong formula is sometimes replaced by $\mathbf{N} \cdot \mathbf{H}$, where $\mathbf{H}$ is a unit vector that bisects the angle between $\mathbf{L}$ and $\mathbf{V}$. Show that $\mathbf{R} \cdot \mathbf{V} \neq \mathbf{N} \cdot \mathbf{H}$.

SOLUTION

Let $\mathbf{L}$, $\mathbf{N}$, and $\mathbf{V}$ be coplanar. Since $\mathbf{L}$, $\mathbf{N}$, and $\mathbf{R}$ must be coplanar from the laws of optics, and $\mathbf{L}$, $\mathbf{V}$, and $\mathbf{H}$, the bisector of the angle between $\mathbf{L}$ and $\mathbf{V}$, must also be coplanar, all the vectors are coplanar (see Fig. 11-18). From the angles shown on the left half of the drawing we have

$$\beta = \theta + \alpha$$

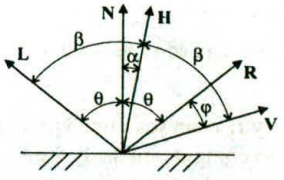


Fig. 11-18

From the angles shown on the right half of the drawing we have

$$\alpha + \beta = \theta + \varphi$$

Combining the two expressions we get

$$\varphi = 2\alpha$$

Unless $\varphi = \alpha = 0$, we have $\varphi \neq \alpha$ and $\cos(\varphi) \neq \cos(\alpha)$. Since $\mathbf{R} \cdot \mathbf{V} = \cos(\varphi)$ and $\mathbf{N} \cdot \mathbf{H} = \cos(\alpha)$, we have

$$\mathbf{R} \cdot \mathbf{V} \neq \mathbf{N} \cdot \mathbf{H}$$

11.12 The color of an object is largely determined by its diffuse reflection coefficient. If $k_d = (0.8, 0.4, 0)$ and the light is blue, what is the color of the object?

SOLUTION

Black, since the object does not reflect blue light and there is no red and green light for it to reflect.

11.13 Refer to Prob. 11.12. What if we use magenta light?

SOLUTION

Red, since the object only reflects the red component of the magenta light.

11.14 When a light source is relatively far away from a polyhedron, the diffuse reflection determined by the Phong formula varies little within each polygonal face. Why?

SOLUTION

All points on a polygonal face share the same normal vector $\mathbf{N}$. When the light is relatively far away, vector $\mathbf{L}$ (see Fig. 11-6) varies little from one surface point to another. (If the light is very distant, say, the sun,

then **L** becomes a constant vector.) Hence $\mathbf{L} \cdot \mathbf{N}$, the term that determines the diffuse component in the Phong formula varies little within each polygonal face.

11.15 Refer to Fig. 11-9, and show exactly how to compute the color of P using bilinear interpolation.

SOLUTION

Let the coordinates of P, P_1, P_2, P_3, P', and P'' be $(x, y), (x_1, y_1), (x_2, y_2), (x_3, y_3), (x', y')$, and (x'', y''), respectively. Also let the color/intensity values at those points be I, I_1, I_2, I_3, I', and I'', and respectively. We have

$$I' = I_1 \frac{y_2 - y'}{y_2 - y_1} + I_2 \frac{y' - y_1}{y_2 - y_1} \qquad I'' = I_2 \frac{y'' - y_3}{y_2 - y_3} + I_3 \frac{y_2 - y''}{y_2 - y_3}$$

and

$$I = I' \frac{x'' - x}{x'' - x'} + I'' \frac{x - x'}{x'' - x'}$$

For *RGB* colors, these formulas are applied to the values of each color component.

11.16 Suppose point P_1 with intensity I_1 is on scan line y_1 and point P_2 with intensity I_2 is on scan line y_2. Find an incremental formula to compute intensity values I' for all the scan lines between P_1 and P_2 using linear interpolation in the y direction.

SOLUTION

Let P_1 be the starting point, then the intensity change from one scan line to the next is

$$\Delta I = (I_2 - I_1)/(y_2 - y_1)$$

Hence

$$I'_1 = I_1 \qquad \text{and} \quad I'_i = I'_{i-1} + \Delta I \qquad \text{where } i = 2, \ldots, y_2 - y_1$$

11.17 Refer to Prob. 11.16. If point P_1 on line 5 has an *RGB* color (1, 0.5, 0) and point P_2 on line 15 has an *RGB* color (0.2, 0.5, 0.6), what is the color for line 8?

SOLUTION

Since

$$\Delta R = (0.2 - 1)/(15 - 5) = -0.08$$
$$\Delta G = (0.5 - 0.5)/(15 - 5) = 0$$
$$\Delta B = (0.6 - 0)/(15 - 5) = 0.06$$

we have $(1 + 3 \times (-0.08), 0.5 + 3 \times 0, 0 + 3 \times 0.06) = (0.76, 0.5, 0.18)$ as the color for line 8.

11.18 Refer to Fig. 11-10(*a*). The vertices are shared by the rectangles that approximate the cylinder and the two polygons that represent the top and bottom of the object. Does each vertex have a unique normal vector?

SOLUTION

No. When a vertex is used to describe the top or bottom face, its normal vector is the normal of the corresponding face. When it is used to define the rectangle mesh, its normal vector is perpendicular to the cylinder as shown in the figure.

11.19 Refer to Fig. 11-11 and see Prob. 11.15. If the two normal vectors at P' and P'' are

$$\mathbf{N}' = -\frac{\sqrt{2}}{2}\mathbf{I} + \frac{\sqrt{2}}{2}\mathbf{J} \qquad \mathbf{N}'' = -\frac{\sqrt{2}}{2}\mathbf{I} + \frac{\sqrt{2}}{2}\mathbf{K}$$

and point P is half way between P' and P'', what is $\mathbf{N}$?

SOLUTION

Since P is in the middle of P' and P'', the last formula in Prob. 11.15 for linear interpolation in the x direction becomes

$$I = \frac{I'}{2} + \frac{I''}{2}$$

Use it to interpolate each component of $\mathbf{N}'$ and $\mathbf{N}''$:

$$\left(-\frac{\sqrt{2}}{4} - \frac{\sqrt{2}}{4}\right)\mathbf{I} + \frac{\sqrt{2}}{4}\mathbf{J} + \frac{\sqrt{2}}{4}\mathbf{K} = -\frac{\sqrt{2}}{2}\mathbf{I} + \frac{\sqrt{2}}{4}\mathbf{J} + \frac{\sqrt{2}}{4}\mathbf{K}$$

The magnitude of this vector is

$$\sqrt{\left(-\frac{\sqrt{2}}{2}\right)^2 + \left(\frac{\sqrt{2}}{4}\right)^2 + \left(\frac{\sqrt{2}}{4}\right)^2} = \sqrt{\frac{2}{4} + \frac{2}{16} + \frac{2}{16}} = \frac{\sqrt{3}}{2}$$

Use it to normalize the interpolated vector and get

$$\mathbf{N} = -\frac{\sqrt{6}}{3}\mathbf{I} + \frac{\sqrt{6}}{6}\mathbf{J} + \frac{\sqrt{6}}{6}\mathbf{K}$$

11.20 Show that when averaging or interpolating normal vectors we will get incorrect result if the vectors are not unit vectors (or vectors of equal magnitude).

SOLUTION

Consider $\mathbf{N}_1 = \mathbf{I}$ and $\mathbf{N}_2 = \mathbf{J}$: the average of the two is $\mathbf{I} + \mathbf{J}$, which bisects the 90° angle between $\mathbf{I}$ and $\mathbf{J}$. We can normalize it to get

$$\mathbf{N} = \frac{\sqrt{2}}{2}\mathbf{I} + \frac{\sqrt{2}}{2}\mathbf{J}$$

Now if we set $\mathbf{N}_2 = 2\mathbf{J}$, which has the same direction as $\mathbf{J}$, the average of $\mathbf{N}_1$ and $\mathbf{N}_2$ becomes $\mathbf{I} + 2\mathbf{J}$. Clearly, the direction of this vector differs from that of $\mathbf{N}$.

As for interpolation we can see from Prob. 11.19 that to compute a vector for the point halfway between two given points we apply

$$I = \frac{I'}{2} + \frac{I''}{2}$$

to each vector component. For $\mathbf{N}_1 = \mathbf{I}$ and $\mathbf{N}_2 = \mathbf{J}$, this produces

$$\tfrac{1}{2}\mathbf{I} + \tfrac{1}{2}\mathbf{J}$$

which is, after normalization, the same as $\mathbf{N}$ above. But, if we use $\mathbf{N}_2 = 2\mathbf{J}$, the result of interpolation becomes

$$\tfrac{1}{2}\mathbf{I} + \mathbf{J}$$

which has the direction of $\mathbf{I} + 2\mathbf{J}$, not the direction of $\mathbf{N}$.

11.21 Consider the texture-blending formula $(1 - k)C + kC'$ where $0 \leq k \leq 1.0$, C is the original color of the object, and C' is the color of the texture map. Describe in simple words the results that correspond to the two extreme cases: $k = 0$ and $k = 1$.

SOLUTION

The case $k = 0$ means no texture at all, whereas the case $k = 1$ means painting the texture over the original shading of the object.

11.22 When we use the logical-operation AND to combine the original color of the object and the color of the texture map, a texture area with magenta shades will appear unaltered if the original color is white, but it will take on red shades if the original is yellow. Why?

SOLUTION

Various shades of magenta can be described by *RGB* color vectors in the form of $(m, 0, m)$. The result of white (1, 1, 1) AND $(m, 0, m)$ is $(m, 0, m)$, whereas the result of yellow (1, 1, 0) AND $(m, 0, m)$ is $(m, 0, 0)$, which represents shades of red.

11.23 See Fig. 11-19, and find the linear functions that map the normalized image onto a square area of 50×50 in the middle of the front face of the cubic object.

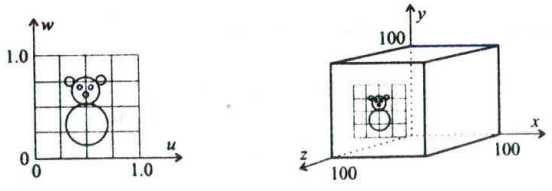


Fig. 11-19

SOLUTION

First find a parametric representation for the target area

$$\begin{cases} x = \theta & 25 \leq \theta \leq 75 \\ y = \varphi & 25 \leq \varphi \leq 75 \\ z = 100 \end{cases}$$

Note the relationship between the corner points

$$\begin{aligned} u = 0, \quad w = 0 \rightarrow \theta = 25, \quad \varphi = 25 \\ u = 1, \quad w = 0 \rightarrow \theta = 75, \quad \varphi = 25 \\ u = 0, \quad w = 1 \rightarrow \theta = 25, \quad \varphi = 75 \\ u = 1, \quad w = 1 \rightarrow \theta = 75, \quad \varphi = 75 \end{aligned}$$

Substitute these into $\theta = Au + B$ and $\varphi = Cw + D$, we get

$$A = 50, \quad B = 25, \quad C = 50, \quad \text{and} \quad D = 25$$

Hence the mapping functions are

$$\theta = 50u + 25 \quad \text{and} \quad \varphi = 50w + 25$$

The inverse mapping functions are

$$u = \frac{\theta - 25}{50} \quad \text{and} \quad w = \frac{\varphi - 25}{50}$$

11.24 See Fig. 11-20. Find the linear functions that map the normalized grid pattern onto the bottom portion ($\pi/4 \le \varphi \le \pi/2$) of the spherical surface in the first octant.

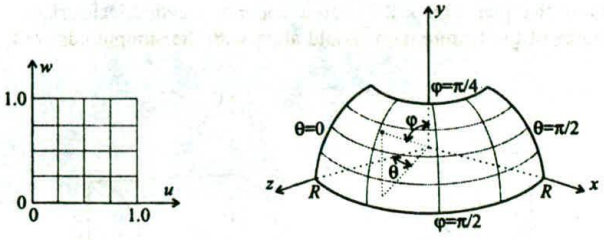


Fig. 11-20

SOLUTION

The parametric representation for the target area is

$$\begin{cases} x = R\sin(\theta)\sin(\varphi) & 0 \le \theta \le \pi/2 \\ y = R\cos(\varphi) & \pi/4 \le \varphi \le \pi/2 \\ z = R\cos(\theta)\sin\varphi) \end{cases}$$

Note the relationship between the corner points

$$\begin{aligned} &u = 0, \quad w = 0 \to \theta = 0, \quad \varphi = \pi/2 \\ &u = 1, \quad w = 0 \to \theta = \pi/2, \quad \varphi = \pi/2 \\ &u = 0, \quad w = 1 \to \theta = 0, \quad \varphi = \pi/4 \\ &u = 1, \quad w = 1 \to \theta = \pi/2, \quad \varphi = \pi/4 \end{aligned}$$

Substitute these into $\theta = Au + B$ and $\varphi = Cw + D$, we get

$$A = \pi/2, \quad B = 0, \quad C = -\pi/4, \quad \text{and} \quad D = \pi/2$$

Hence the mapping functions are

$$\theta = \frac{\pi}{2}u \quad \text{and} \quad \varphi = \frac{\pi}{2} - \frac{\pi}{4}w$$

The inverse mapping functions are

$$u = \frac{\theta}{\pi/2} \quad \text{and} \quad w = \frac{\pi/2 - \varphi}{\pi/4}$$

Supplementary Problems

11.25 Given the distribution function $P(\lambda)$ in Fig. 11-3, derive a formula to calculate saturation.

11.26 Why everything looks gray or black in a dark room where we can barely see?

11.27 Can we use Z to convert from chromaticity coordinates (x, y) back to a specific color in the XYZ color space?

11.28 What's the difference between the Y in CMY and the Y in YIQ?

11.29 Consider a sphere of radius R, centered at (x_c, y_c, z_c). What is the normal vector **N** at point (x, y, z) on the sphere?

11.30 Find the linear functions that map a 1.0 × 2.0 texture map onto a cylindrical surface in the first octant (see Fig. 11-21) . The short sides of the texture map should align with the straight edges of the surface.

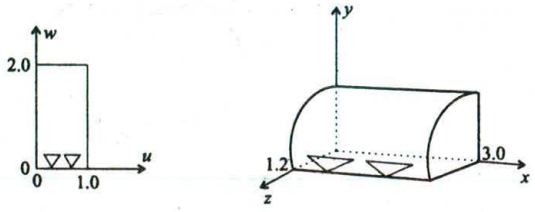


Fig. 11-21

CHAPTER 12

Ray Tracing

Ray tracing is a global illumination model that accounts for the transport of light energy beyond the direct/local contribution from the light sources. Its operation is largely based on the principles of geometric optics. By tracing the path of light rays it is able to integrate the tasks of shading, shadow generation, hidden surface removal, projection, and scan conversion into one single computational process.

In this chapter we first introduce the fundamental concepts that underlie this elegant approach (Sect. 12.1). We then discuss the basic ray-tracing algorithm (Sect. 12.2), the parametric vector representation of a ray (Sect. 12.3), and the mathematics of ray–surface intersection (Sect. 12.4). We also present techniques for improving execution efficiency (Sect. 12.5), anti-aliasing (Sect. 12.6), and achieving some desired visual effects (Sect. 12.7).

12.1 THE PINHOLE CAMERA

In theory one can take perfect pictures using a pinhole camera—a closed box with a tiny hole in the center of the front panel (see Fig. 12-1). The hole is so small that only one light ray passing through it can strike a particular point on the negative, which is mounted on the inside of the back panel. As light rays from across the scene expose the negative by hitting their respective target precisely, a sharp image depicting the scene emerges.

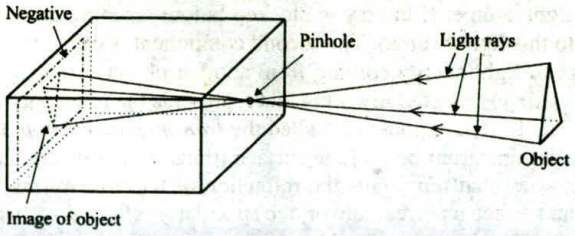


Fig. 12-1 A pinhole camera.

Now we place a screen between the pinhole and the object (see Fig. 12-2), causing the light rays to intersect the screen. If we can record each light ray as it intersects the screen, we will have a perfect picture of the object on the screen. Or, to put this in terms of image synthesis, if the screen resolution is arbitrarily high and we can set the pixel at each intersection point to match the color of the corresponding light ray, then the image will be indistinguishable from the real scene when viewed from the position of the pinhole.

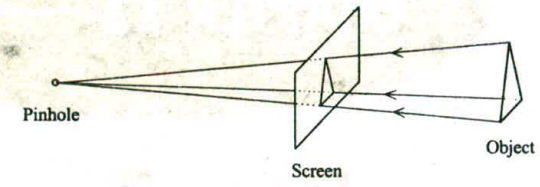


Fig. 12-2

12.2 A RECURSIVE RAY-TRACER

In order to construct an image based on the pinhole-camera model we need to determine the light rays that are responsible for the pixels. We do so by following their path in the opposite direction, i.e., from the viewpoint through the center of each pixel into the scene (see Fig. 12-3). These are called the *primary rays*. If a primary ray does not intersect any object in the scene, then the corresponding pixel is simply set to some background color.

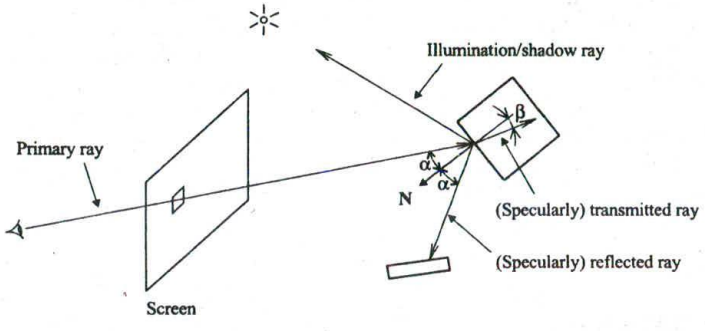


Fig. 12-3 Ray tracing.

On the other hand, if a primary ray intersects an object, then the color of the corresponding pixel is determined by the surface shading of the object at the intersection point. Several *secondary rays* are used to compute the three components of this surface shading. The first component is called the *local contribution*, which refers to the direct contribution from the light source(s). We send a *shadow ray* or *illumination ray* from the surface point to a light source. If the ray is blocked before reaching the light source, the surface point is in shadow (relative to this light source). The second component is called the *reflected contribution*, which refers to the reflection of light energy coming from another object surface (inter-object reflection). This is determined by a (*specularly*) *reflected ray*, a ray that represents the reflection of the primary ray with respect to normal vector **N**. The third component is called the *transmitted contribution*, which refers to the transmission of light energy coming from behind the surface (transparent object). This is determined by a (*specularly*) *transmitted ray*, a ray that represents the refraction of the primary ray with respect to **N**.

Note that, were the object surface a perfect mirror, the specularly reflected ray would represent the sole direction of light reflected towards the viewpoint. Also, if the object were made of perfectly homogeneous material, the specularly transmitted ray would represent the sole direction of light refracted towards the viewpoint.

Snell's law determines the relationship between angles α and β (see Fig. 12-3):

$$\frac{\sin(\alpha)}{\sin(\beta)} = \frac{\eta_2}{\eta_1}$$

where η_1 is the refraction index of the medium in which α is defined and η_2 is the refraction index of the medium in which β is defined. Although the refraction index of a medium is a function of temperature and wavelength, we can often use such values as 1.0 for air, 1.3 for water, and 1.5 for glass.

Employing the Phong formula to describe the local contribution, we express the intensity I of the light energy represented by the primary ray as

$$I = I_a k_a + \sum_{i=1}^{n} I_{p_i}(k_d \mathbf{L}_i \cdot \mathbf{N} + k_s(\mathbf{R}_i \cdot \mathbf{V})^k) + k_r I_r + k_t I_t$$

where the ambient term now represents the portion of inter-object reflection that is not accounted for by the term $k_r I_r$, which describes the reflected contribution as the product of the reflection coefficient k_r and the intensity I_r of the reflected ray; the last term $k_t I_t$ describes the transmitted contribution as the product of the transmission coefficient k_t and the intensity I_t of the transmitted ray. Both k_r and k_t are in the range of [0, 1.0]. I_r and I_t are obtained recursively by treating the reflected ray and the transmitted ray as a primary ray. In other words, we calculate I_r by evaluating the above equation at the closest surface the reflected ray intersects, in exactly the same way as the primary ray is handled. A pseudo-code description of the basic ray-tracing algorithm is as follows:

```
rayTrace{ray, depth, color}
{
  determine closest intersection of ray with an object;
  if (no intersection) color = background;
  else {
    color = local contribution;
    if (depth > 1) {
      calculate reflected ray;
      rayTrace(reflected ray, depth-1, ref_color);
      calculate transmitted ray;
      rayTrace(transmitted ray, depth-1, trans_color);
      color = color + k_r · ref_color + k_t · trans_color;
    }
  }
}
```

This procedure takes a ray and a depth value that is greater than or equal to 1 as input, and returns a color as output (typically an *RGB* vector, the above formula is used to compute the intensity of each individual color component). It treats the given ray as a primary ray and (when depth > 1) makes recursive calls to obtain the color (ref_color) of the reflected ray and the color (trans_color) of the transmitted ray.

Figures 12-4, 12-5, and 12-6 show the results of ray-tracing a scene consisting of three opaque spheres. The smaller one in the front is dull whereas the two larger ones in the back are shiny. The first image (depth = 1) is obtained using only the primary ray and shadow ray. The second image (depth = 2) depicts the effect of adding reflected contribution (one level beyond the primary ray). The third image (depth = 3) shows the cumulative effect of two levels of reflected contribution. In Fig. 12-7 (also depth = 3), the dull sphere is replaced by a shiny glass sphere (a check-board floor is also added) to illustrate the combined effect of all three shading components.

12.3 PARAMETRIC VECTOR REPRESENTATION OF A RAY

A ray is not a vector although it may look like one (see Fig. 12-3). The difference between the two is that, while a vector is defined by its direction and magnitude, a ray is determined by its direction and starting point.

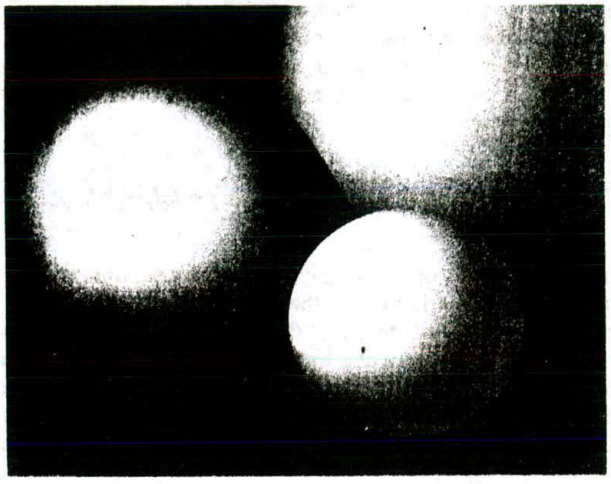


Fig. 12-4

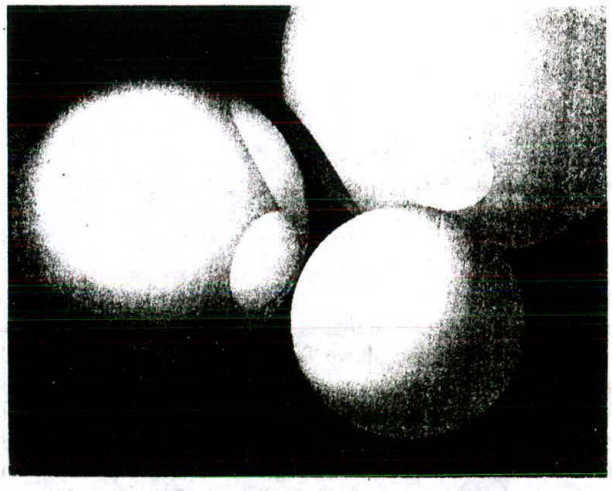


Fig. 12-5

We may represent a ray in terms of two vectors: **s** to specify its starting point, and **d** to describe its direction (see Fig. 12-8). These two vectors are used to provide a parametric vector representation for the ray:

$$\mathbf{r}(t) = \mathbf{s} + t\mathbf{d} \qquad (0 \leq t)$$

where $\mathbf{r}(t)$ denotes a family of vectors. When the tails of these vectors are placed at the origin, their heads make up the ray.

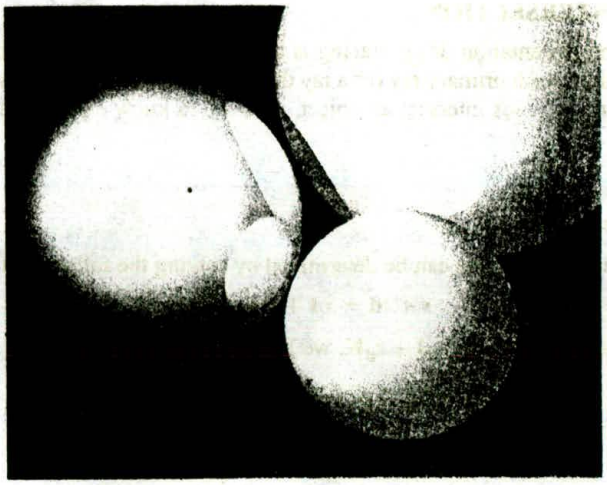


Fig. 12-6

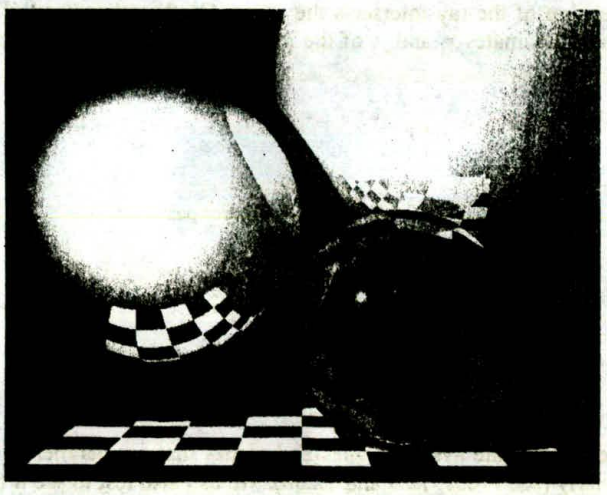


Fig. 12-7

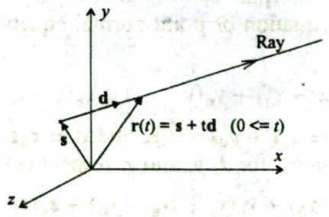


Fig. 12-8 Vector representation of a ray.

12.4 RAY–SURFACE INTERSECTION

A critical step in the implementation of ray tracing is to determine whether a ray intersects an object. This operation is performed for each primary ray (or a ray that is treated as a primary ray in a recursive call) and shadow ray. If a primary ray does intersect an object, we need to know exactly where the intersection point is.

Coordinate System Plane

Intersection of a ray with the xy plane can be determined by solving the following for t (see Fig. 12-9):

$$\mathbf{s} + t\mathbf{d} = x_i\mathbf{I} + y_i\mathbf{J}$$

With $\mathbf{s} = x_s\mathbf{I} + y_s\mathbf{J} + z_s\mathbf{K}$ and $\mathbf{d} = x_d\mathbf{I} + y_d\mathbf{J} + z_d\mathbf{K}$, we have

$$\begin{cases} x_s + tx_d = x_i \\ y_s + ty_d = y_i \\ z_s + tz_d = 0 \end{cases}$$

When $z_d = 0$, the ray is parallel to the plane (no intersection). When $z_s = 0$, the ray originates from the plane (no intersection). Otherwise, we calculate t using the third equation

$$t = -\frac{z_s}{z_d}$$

If $t < 0$, the negative extension of the ray intersects the plane. On the other hand, if $t > 0$, the ray itself intersects the plane and the coordinates x_i and y_i of the intersection point can be calculated from the first two equations.

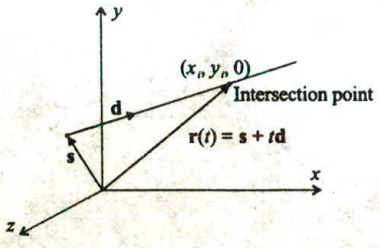


Fig. 12-9

Intersection with other coordinate system planes and planes that are parallel to a coordinate system plane can be handled similarly (see Probs. 12.9 and 12.30). We can also test to see if the intersection point is within a region (see Probs. 12.10, 12.11, and 12.12).

Arbitrary Plane

Let $\mathbf{n} = x_n\mathbf{I} + y_n\mathbf{J} + z_n\mathbf{K}$ be the normal vector to an arbitrary plane and $P_0\ (x_0, y_0, z_0)$ a point on the plane (see Fig. 12-10). The vector equation or point-normal equation of the plane is then (from App. 2, Sect. A2.3)

$$x_n(x - x_0) + y_n(y - y_0) + z_n(z - z_0) = 0$$

To determine if a ray $\mathbf{s} + t\mathbf{d}$ ($\mathbf{s} = x_s\mathbf{I} + y_s\mathbf{J} + z_s\mathbf{K}$ and $\mathbf{d} = x_d\mathbf{I} + y_d\mathbf{J} + z_d\mathbf{K}$) intersects the plane, we substitute $x_s + tx_d$, $y_s + ty_d$, and $z_s + tz_d$ for x, y, and z, respectively:

$$x_n(x_s + tx_d - x_0) + y_n(y_s + ty_d - y_0) + z_n(z_s + tz_d - z_0) = 0$$

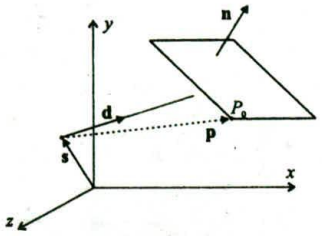


Fig. 12-10

Solving for t we get

$$t = -\frac{x_n(x_s - x_0) + y_n(y_s - y_0) + z_n(z_s - z_0)}{x_n x_d + y_n y_d + z_n z_d}$$

Introducing $\mathbf{p} = (x_0 - x_s)\mathbf{I} + (y_0 - y_s)\mathbf{J} + (z_0 - z_s)\mathbf{K}$, we can express t in vector form:

$$t = \frac{\mathbf{n} \cdot \mathbf{p}}{\mathbf{n} \cdot \mathbf{d}}$$

When $\mathbf{n} \cdot \mathbf{d} = 0$, the ray is parallel to the plane (no intersection). Otherwise, if $t < 0$, the negative extension of the ray intersects the plane; if $t = 0$, the ray originates from the plane. The ray intersects the plane only when $t > 0$.

Sometimes we want to further distinguish the two cases of intersection; intersecting the outside (front) of a plane versus intersecting its inside (back). The former occurs when $\mathbf{n} \cdot \mathbf{d} < 0$ whereas the latter occurs when $\mathbf{n} \cdot \mathbf{d} > 0$.

Sphere

Consider a sphere of radius R centered at (x_c, y_c, z_c) and an arbitrary point (x, y, z) on the sphere (see Fig. 12-11). Define vectors $\mathbf{c} = x_c\mathbf{I} + y_c\mathbf{J} + z_c\mathbf{K}$ and $\mathbf{p} = x\mathbf{I} + y\mathbf{J} + z\mathbf{K}$. The vector equation for the sphere is

$$|\mathbf{p} - \mathbf{c}| = R$$

To determine if a ray $\mathbf{r}(t) = \mathbf{s} + t\mathbf{d}$ intersects the sphere, we substitute $\mathbf{s} + t\mathbf{d}$ and $\mathbf{p}$ and square both sides:

$$|\mathbf{s} + t\mathbf{d} - \mathbf{c}|^2 = R^2$$

Since $\mathbf{v} \cdot \mathbf{v} = |\mathbf{v}|^2$ for any vector $\mathbf{v}$ (see Prob. A2.3), we have

$$\begin{aligned}
|\mathbf{s} - \mathbf{c} + t\mathbf{d}|^2 &= (\mathbf{s} - \mathbf{c} + t\mathbf{d}) \cdot (\mathbf{s} - \mathbf{c} + t\mathbf{d}) \\
&= (\mathbf{s} - \mathbf{c}) \cdot (\mathbf{s} - \mathbf{c} + t\mathbf{d}) + t\mathbf{d} \cdot (\mathbf{s} - \mathbf{c} + t\mathbf{d}) \qquad \text{(Prob. A1.20)} \\
&= (\mathbf{s} - \mathbf{c}) \cdot (\mathbf{s} - \mathbf{c}) + 2(\mathbf{s} - \mathbf{c}) \cdot t\mathbf{d} + t\mathbf{d} \cdot t\mathbf{d} \qquad \text{(Prob. A1.19)} \\
&= |\mathbf{s} - \mathbf{c}|^2 + 2t(\mathbf{s} - \mathbf{c}) \cdot \mathbf{d} + t^2|\mathbf{d}|^2 = R^2
\end{aligned}$$

This is in the form of a quadratic equation in t:

$$At^2 + 2Bt + C = 0$$

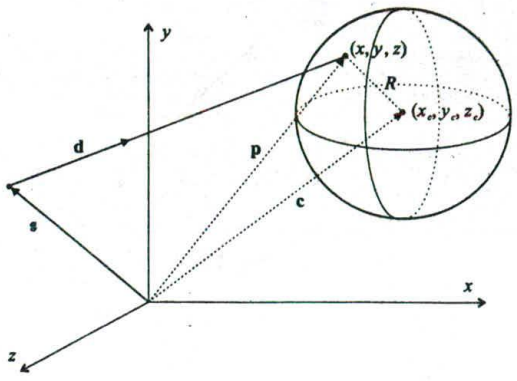


Fig. 12-11

where $A = |\mathbf{d}|^2$, $B = (\mathbf{s} - \mathbf{c}) \cdot \mathbf{d}$, $C = |\mathbf{s} - \mathbf{c}|^2 - R^2$, and the solution is

$$t = \frac{-B \pm \sqrt{B^2 - AC}}{A}$$

with

$$B^2 - AC \begin{cases} < 0 & \text{no intersection} \\ = 0 & \text{ray (or its negative extension) touching sphere} \\ > 0 & \text{two (possible) intersection points} \end{cases}$$

When $B^2 - AC > 0$ we get two t values: t_1 and t_2. If both are less than 0, the negative extension of the ray intersects the sphere (no intersection by the ray). If one of them is 0, the ray starts from a point on the sphere and intersects the sphere only if the other value is positive. If the two values differ in signs, the ray originates from inside the sphere and intersects the sphere once. If both are positive, the ray intersects the sphere twice (first enters and then exits), and the smaller t value corresponds to the intersection point that is closer to the starting point of the ray.

General Implicit Surface

Generally, to determine if a ray $\mathbf{s} + t\mathbf{d}$ ($\mathbf{s} = x_s\mathbf{I} + y_s\mathbf{J} + z_s\mathbf{K}$ and $\mathbf{d} = x_d\mathbf{I} + y_d\mathbf{J} + z_d\mathbf{K}$) intersects a surface that is represented by an implicit equation $F(x, y, z) = 0$, we solve

$$F(x_s + tx_d, y_s + ty_d, z_s + tz_d) = 0$$

for t (see Probs. 12.16, 12.17, and 12.31).

12.5 EXECUTION EFFICIENCY

Ray-tracing is time-consuming largely due to the recursive nature of the algorithm and the demanding task of computing ray–surface intersections. Several techniques are designed to improve execution efficiency by means of object/scene-dependent deployment of system resources.

Adaptive Depth Control

An opaque object ($k_t = 0$) does not transmit light. A dull object ($k_r = 0$) does not produce specular reflection. In these extreme cases there is clearly no need to trace the transmitted/reflected ray, since the transmitted/reflected contribution is zero regardless of the value of I_t/I_r.

Even in a more general situation, say, $k_r > 0$, the cumulative effect of the reflection coefficients of the surfaces along the path of reflection can render the contribution from additional recursive steps negligible. For example (see Fig. 12-12), if the reflection coefficients of the objects along the path of reflection are k_{r1}, k_{r2}, and k_{r3}, respectively, the eventual contribution of I_{r3} to the primary ray is $k_{r1}k_{r2}k_{r3}I_{r3}$. When $k_{r1} = k_{r2} = k_{r3} = 0.1$, this contribution is $0.001I_{r3}$ (negligible in most applications).

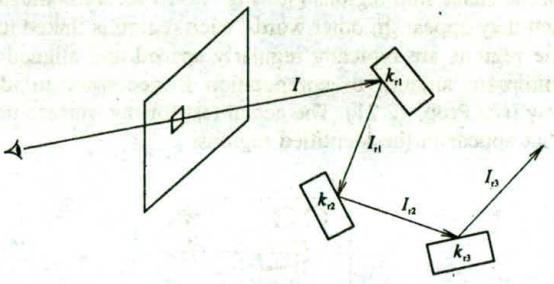


Fig. 12-12

Using adaptive depth control, the ray-tracer keeps track of the cumulative attenuation along the path of reflection (and transmission in a similar manner) and, before working on another reflected ray, compares the current value of cumulative attenuation to a present threshold. It continues its recursive execution to the required depth only if the cumulative attenuation does not fall below the threshold. For the example in Fig. 12-12, if $k_{r1} = k_{r2} = k_{r3} = 0.1$ and the threshold is 0.01, then the ray-tracer will not go beyond the second reflected ray because $k_{r1}k_{r2}k_{r3} < 0.01$.

Bounding Volume Extension

Since a ray travels along a narrow path in one specific direction, it normally misses most of the objects in the scene. This means that an object is more likely to get a "no" answer in a ray–surface-intersection test than a "yes" answer. The purpose of this technique is to identify objects, especially complex objects, that are definitely not intersected by the ray as quickly as possible.

Each object or group of objects in close spatial proximity is surrounded by a capsule/bounding volume (e.g., a sphere, a box, etc.) that permits a simple intersection check. If the ray does not intersect the capsule, it does not intersect the object(s) inside the capsule, and no further testing involving the object(s) is necessary. On the other hand, if the ray does intersect the capsule, further testing involving the enclosed object(s) is necessary in order to reach a conclusion ("yes" or "no").

Hierarchy of Bounding Volumes

The bounding volume technique can be extended to a hierarchy of bounding volumes. For example, if a monitor and a printer are on top of a desk, we may introduce a large bounding volume encompassing all three objects, and three smaller bounding volumes inside the large one for the individual objects, respectively. Only if a ray intersects the large bounding volume it is necessary to go to the next level of the hierarchy to test against the three smaller bounding volumes.

Bounding volume techniques incur two types of overhead: maintaining bounding volumes and testing against bounding volumes. It is often worth the effort when the objects are complex and are not loosely distributed across the scene (see Prob. 12.20).

Spatial Coherence/Spatial Subdivision

This technique is based on the observation that only objects that are in the vicinity of the path of a ray may be intersected by the ray, objects that are away from the path do not have anything to do with the ray (and should be ignored in intersection testing).

We subdivide the scene space into regions (see Fig. 12-13 for a 2D illustration) and associate objects with the regions in which they appear. In other words, each region is linked to a list of objects that appear in that region. Since the regions are typically regularly spaced and aligned with the coordinate system axes/planes, only a minimum amount of computation is necessary to identify the regions that are intersected by a given ray (see Prob. 12.21). The actual test for ray–surface intersection is then performed only with the objects that appear in the identified regions.

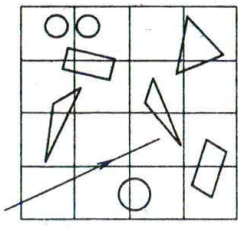


Fig. 12-13

Furthermore, we follow the ray from one region to the next and perform intersection tests as we enter each region (since an object may be associated with more than one region, care should be taken to avoid repeating intersection tests with the same object). Once an intersection point (closest to the starting point of the ray) is identified with the objects in the current region, there is no need to move forward into other regions.

The overhead for spatial subdivision is mainly the need to preprocess the scene in order to assign objects to regions. The technique is complementary to the bounding volume approach in that it is quite capable of dealing with objects that are scattered around the scene.

12.6 ANTI-ALIASING

Ray-tracing depicts a continuous scene by taking discrete samples from the scene. Hence the aliasing artifacts associated with scan-conversion (e.g., jagged edges, dropout of small parts/objects) are also present in ray-traced images. The following are some anti-aliasing techniques that can be incorporated into the ray-tracing process to alleviate the problem.

Supersampling

Each pixel is divided into subpixels (see Chap. 3, Fig. 3-27) and a separate primary ray is sent and traced through the center of each subpixel. The color/intensity values of the primary rays for the subpixels are then combined (averaged) to produce a value for the parent pixel.

Adaptive Supersampling

Supersampling often drastically increases the computational burden of ray tracing. In this approach we send one ray through the center of a pixel and four additional rays through its corners. If the five rays return similar colors the pixel will not be subdivided (it probably corresponds to a smoothly shaded area in the scene). Only if the returned colors are sufficiently different do we subdivide the pixel (it probably covers both sides of an edge) into 2×2 subpixels. Each subpixel is then handled in the same way, and the process terminates when a preset level of subdivision is reached.

Stochastic Supersampling

The effect of supersampling can often be enhanced with stochastic or distributed ray-tracing. In this approach we deviate from using the fixed (sub)pixel grid by scattering the rays evenly across the pixel area in a random fashion. A typical way to achieve this is to displace each ray from its normal position in the grid (see Prob. 12.24).

12.7 ADDITIONAL VISUAL EFFECTS

Various techniques have been developed to achieve certain desirable visual effects. Some methods below are applications of *distributed ray-tracing*, which means that we scatter rays around with respect to a certain parameter in the ray-tracing process (recall that in stochastic supersampling we displace rays from their normal position in the pixel grid).

Environment Mapping

A shiny (mirror-like) object reflects the surrounding environment. Instead of ray-tracing the three-dimensional scene to obtain the global reflection, we may map a picture of the environment onto the object (see Fig. 12-14). The object is typically placed in the middle of an enclosure such as a cube, cylinder, or sphere, with the environment map attached to the inside of the enclosing surface (facing the object). The color of a pixel is then a function (e.g., average by supersampling) of the area in the environment map that corresponds to the pixel.

Clearly this is only an approximation to ray-tracing the three-dimensional scene. The quality of the mapped reflection is dependent on the size (relative to the enclosure) and shape of the object. However,

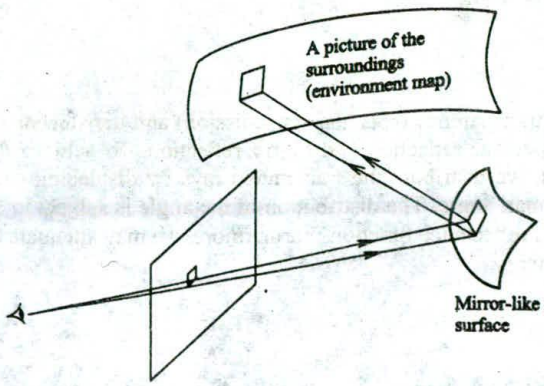


Fig. 12-14

environment mapping is often faster, and it is particularly useful in creating the illusion of a computer-generated shiny object amidst a real-world scene (which of course cannot be ray-traced).

Soft Shadow

Ray-traced images of scene involving fixed point-lights are characterized by the harsh edges of the shadow areas. However real lights are not mathematical points and they cast *soft shadows*, which consist of an *umbra* area surrounded by a *penumbra* area (see Fig. 12- 15).

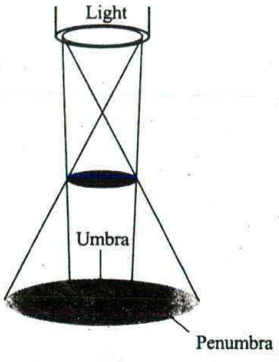


Fig. 12-15

In order to produce soft shadows we model a light source by a region, called an *area light*. The area is subdivided into subareas or zones. Shadow rays are then distributed to these zones via random selection (with equal probability for each zone or weights that favor zones that correspond to higher intensity values).

Blurry Reflection

To get a blurry reflection of the surroundings on a glossy (not mirror-like) object, we distribute the reflected rays. This can be done by displacing a reflected ray from the position of its mirror reflection by a small angle. The distribution of the angle is subject to the same bell-shaped reflectance function that governs specular highlights (see Fig. 11-6).

Translucency

The difference between transparency (specular transmission) and translucency is somewhat analogous to the difference between specular reflection and blurry reflection. To achieve the effect of light going through translucent material, we distribute the transmitted rays by displacing each from the position of specular transmission by a small angle. The distribution of the angle is subject to a transmittance function similar to the aforementioned reflectance function. Furthermore, we may attenuate the transmitted intensity to account for the loss of energy.

Motion Blur

A fast-moving object tends to look blurry in a photograph (especially if the photo was taken with a low shutter speed). To mimic this phenomenon we distribute rays over time. In other words, we predetermine

the path or a series of positions of a moving object based on the characteristics of the movement. We then displace the object along its path or between positions as each ray is traced.

Solved Problems

12.1 Describe how hidden surface removal and projection are integrated into the ray-tracing process.

SOLUTION

Each primary ray plays the role of a projector that maps a surface point P to its image P' in the view plane. If all primary rays emanate from a view point C we have the effect of perspective projection. If all primary rays are parallel to each other we have the effect of parallel projection. Furthermore, if a primary ray intersects several object surfaces, only the surface that is the closest to the view plane is chosen to determine the color of the corresponding pixel; all others are effectively treated as hidden surfaces and removed (not displayed).

12.2 Name the three components of surface shading and the secondary ray for computing each.

SOLUTION

Local contribution (shadow ray), reflected contribution (specularly reflected ray), and transmitted contribution (specularly transmitted ray).

12.3 The refraction index of a medium is the ratio of the speed of light in vacuum to the speed of light in that medium. Rewrite Snell's law in terms of the speed of light in the two participating media.

SOLUTION

Let c be the speed of light in vacuum, c_1 be the speed of light in the medium whose refraction index is η_1, and c_2 be the speed of light in the medium whose refraction index is η_2. We have $\eta_1 = c/c_1$ and $\eta_2 = c/c_2$. Hence Snell's law can be written as

$$\frac{\sin(\alpha)}{\sin(\beta)} = \frac{\eta_2}{\eta_1} = \frac{c/c_2}{c/c_1} = \frac{c_1}{c_2}$$

12.4 What is the difference between a vector and a ray?

SOLUTION

A vector is defined by its direction and magnitude, whereas a ray is determined by its direction and starting point.

12.5 A ray is represented by $\mathbf{r}(t) = \mathbf{s} + t\mathbf{d}$ where $\mathbf{s} = 2\mathbf{I} + \mathbf{J} - 3\mathbf{K}$ and $\mathbf{d} = \mathbf{I} + 2\mathbf{K}$. Find the coordinates of the points on the ray that correspond to $t = 0$, 1, 2.5, and 3, respectively.

SOLUTION

$$\mathbf{r}(0) = (2+0)\mathbf{I} + \mathbf{J} + (-3+0)\mathbf{K} = 2\mathbf{I} + \mathbf{J} - 3\mathbf{K} \rightarrow (2, 1, -3)$$
$$\mathbf{r}(1) = (2+1)\mathbf{I} + \mathbf{J} + (-3+2)\mathbf{K} = 3\mathbf{I} + \mathbf{J} - \mathbf{K} \rightarrow (3, 1, -1)$$
$$\mathbf{r}(2.5) = (2+2.5)\mathbf{I} + \mathbf{J} + (-3+5)\mathbf{K} = 4.5\mathbf{I} + \mathbf{J} + 2\mathbf{K} \rightarrow (4.5, 1, 2)$$
$$\mathbf{r}(3) = (2+3)\mathbf{I} + \mathbf{J} + (-3+6)\mathbf{K} = 5\mathbf{I} + \mathbf{J} + 3\mathbf{K} \rightarrow (5, 1, 3)$$

12.6 Let the viewpoint be at (a, b, c) and the center of a pixel at (x, y, z). Find vectors $\mathbf{s}$ and $\mathbf{d}$ to represent the corresponding primary ray.

SOLUTION

$$\mathbf{s} = a\mathbf{I} + b\mathbf{J} + c\mathbf{K}$$
$$\mathbf{d} = (x - a)\mathbf{I} + (y - b)\mathbf{J} + (z - c)\mathbf{K}$$

12.7 Lines in two-dimensional space can be represented by either the algebraic equation $y = mx + b$ or the parametric vector equation $\mathbf{L}(t) = \mathbf{s} + t\mathbf{d}$, where $-\infty < t < +\infty$. For $\mathbf{s} = \mathbf{I} + \mathbf{J}$ and $\mathbf{d} = \mathbf{I} - \mathbf{J}$, find the equivalent algebraic representation.

SOLUTION 1

Since $\mathbf{L}(t) = \mathbf{s} + t\mathbf{d} = (1 + t)\mathbf{I} + (1 - t)\mathbf{J}$, we have

$$x = 1 + t \quad \text{and} \quad y = 1 - t$$

Hence $x + y = 2$, or $y = -x + 2$.

SOLUTION 2

Find two points on the line

$$\mathbf{L}(0) = \mathbf{s} = \mathbf{I} + \mathbf{J} \rightarrow (1, 1)$$
$$\mathbf{L}(1) = \mathbf{s} + \mathbf{d} = 2\mathbf{I} \rightarrow (2, 0)$$

We have $(y - 1)/(x - 1) = (0 - 1)/(2 - 1)$, i.e., $y = -x + 2$.

12.8 Refer to Fig. 12-16(a), where $\mathbf{d}$ represents the direction of the primary ray, $\mathbf{t}$ the direction of the transmitted ray, and $\mathbf{n}$ the normal vector at the intersection point. Express $\mathbf{t}$ in terms of $\mathbf{d}$, $\mathbf{n}$, and the two angles α and β.

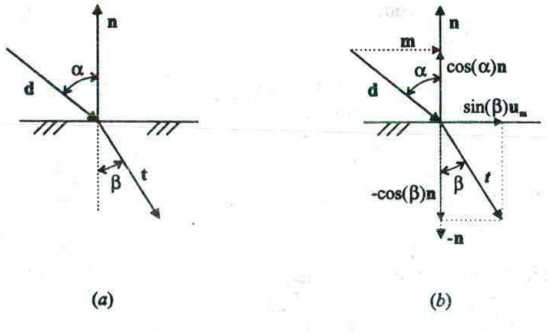


Fig. 12-16

SOLUTION

Let $\mathbf{d}$, $\mathbf{t}$, $\mathbf{n}$ be unit vectors [see Fig. 12-16(b)]. We have

$$\begin{cases} \mathbf{m} = \mathbf{d} + \cos(\alpha)\mathbf{n} \\ |\mathbf{m}| = \sin(\alpha) \end{cases}$$

and

$$\begin{aligned}\mathbf{t} &= \sin(\beta)\mathbf{u}_{\mathbf{m}} - \cos(\beta)\mathbf{n} \\ &= \frac{\sin(\beta)}{\sin(\alpha)}(\mathbf{d} + \cos(\alpha)\mathbf{n}) - \cos(\beta)\mathbf{n}\end{aligned}$$

12.9 Determine if a ray intersects a plane that is parallel to the xy plane.

SOLUTION

The equation for such a plane is $z = c$ where c is a constant. If a ray does intersect the plane, the intersection point is at (x_i, y_i, c). In order to find x_i and y_i we solve the following for t:

$$\mathbf{s} + t\mathbf{d} = x_i\mathbf{I} + y_i\mathbf{J} + c\mathbf{K}$$

With $\mathbf{s} = x_s\mathbf{I} + y_s\mathbf{J} + z_s\mathbf{K}$ and $\mathbf{d} = x_d\mathbf{I} + y_d\mathbf{J} + z_d\mathbf{K}$, we have

$$\begin{cases} x_s + tx_d = x_i \\ y_s + ty_d = y_i \\ z_s + tz_d = c \end{cases}$$

When $z_d = 0$, the ray is parallel to the plane (no intersection). When $z_s = c$, the ray originates from the plane (no intersection). Otherwise, we calculate t using the third equation

$$t = \frac{c - z_s}{z_d}$$

If $t < 0$, the negative extension of the ray intersects the plane. On the other hand, if $t > 0$, the ray itself intersects the plane and the coordinates x_i and y_i of the intersection point can be calculated from the first two equations.

12.10 Determine if a ray intersects a rectangular region defined by $x_{\min}$, $x_{\max}$, $y_{\min}$ and $y_{\max}$ in the xy plane.

SOLUTION

First determine if the ray intersects the xy plane (see Sect. 12.4 under "Coordinate System Plane"). If not, the ray does not intersect the region. Otherwise, find the coordinates x_i and y_i of the intersection point. If $x_{\min} \le x_i \le x_{\max}$ and $y_{\min} \le y_i \le y_{\max}$ the ray intersects the region; otherwise it does not.

12.11 Determine if a ray intersects a triangular (or convex polygonal) region in the xy plane.

SOLUTION

First determine if the ray intersects the xy plane (see Sect. 12.4 under "Coordinate System Plane"). If not, the ray does not intersect any region in the plane. Otherwise, find the coordinates x_i and y_i of the intersection point. The point is inside a triangular or convex polygonal region if it is on/inside all bounding edges of the region.

Now focus on just the xy plane (ignore z). To test if the intersection point (x_i, y_i) is inside or outside an edge in the plane, we choose a point on the edge (e.g., one of its endpoints) and define an outward normal vector $\mathbf{n}$ that points to the outside of the edge (see Fig. 12-17). We also use the chosen point to define vector $\mathbf{v}$, and the intersection point to define vector $\mathbf{v}_i$. We have

$$\mathbf{n} \cdot (\mathbf{v}_i - \mathbf{v}) \begin{cases} > 0 & (x_i, y_i) \text{ is outside the edge} \quad (\theta < 90°) \\ = 0 & (x_i, y_i) \text{ is on the edge} \quad (\theta = 90°) \\ < 0 & (x_i, y_i) \text{ is inside the edge} \quad (\theta > 90°) \end{cases}$$

12.12 A unit square is placed in the xy plane with one corner at the origin and the diagonal corner on the positive y axis [see Fig. 12-18(a)]. Determine if a ray emanating from $(0, 1, 2)$ going in the direction of the negative z axis intersects the square.

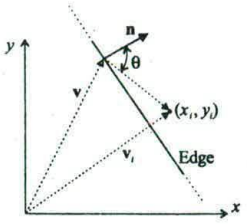


Fig. 12-17

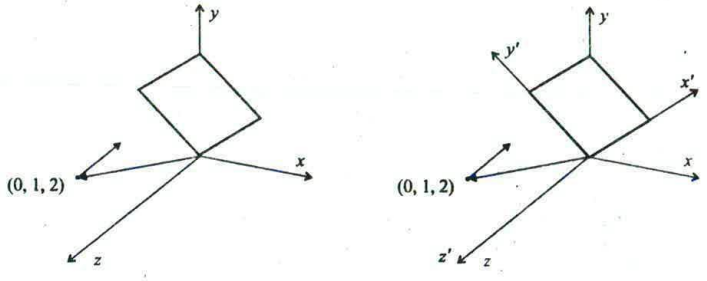


Fig. 12-18

SOLUTION

From the given condition we have

$$\mathbf{s} = \mathbf{J} + 2\mathbf{K} \qquad \text{and} \qquad \mathbf{d} = -\mathbf{K}$$

Since $z_s = 2 \neq 0$, $z_d = -1 \neq 0$, and $t = -z_s/z_d = 2 > 0$, the ray intersects the xy plane (see Sect. 12.4 under "Coordinate System Plane"). Using $x_s = 0$, $x_d = 0$, $y_s = 1$, and $y_d = 0$, we get

$$x_i = 0 + 2 \times 0 = 0 \qquad \text{and} \qquad y_i = 1 + 2 \times 0 = 1$$

To determine if the intersection point is inside the square, we may follow the solution for Prob. 12.10. Alternatively, we may perform a coordinate transformation from system xyz to $x'y'z'$ in order to align the x' and y' axes with the sides of the square [see Fig. 12-18(*b*)]. This requires a 45° rotation of the xyz system with respect to the z axis (or the origin if we ignore the z dimension). The square is now bounded by $x_{\min} = 0$, $x_{\max} = 1$, $y_{\min} = 0$, and $y_{\max} = 1$ in the new coordinate system, where the coordinate of the intersection point are (see Sect. 4.2)

$$x_i' = x_i \cos(45°) + y_i \sin(45°) = \frac{\sqrt{2}}{2}$$

$$y_i' = -x_i \sin(45°) + y_i \cos(45°) = \frac{\sqrt{2}}{2}$$

Since

$$x_{\min} \leq x_i' \leq x_{\max} \qquad \text{and} \qquad y_{\min} \leq y_i' \leq y_{\max}$$

the ray does intersect the square.

12.13 Let $\mathbf{n} = \mathbf{I} + \mathbf{J} + 2\mathbf{K}$ be the normal vector of a plane that passes through point P_0 $(1, 1, 0)$. Determine if a ray with $\mathbf{s} = -2\mathbf{I} + \mathbf{J} + 2\mathbf{K}$ and $\mathbf{d} = \mathbf{I} - \mathbf{K}$ intersects the plane.

SOLUTION

Refer to Sect. 12.4 under arbitrary plane. Since

$$\mathbf{n} \cdot \mathbf{d} = 1 \times 1 + 1 \times 0 + 2 \times (-1) = -1 < 0$$

the ray intersects the plane. We introduce

$$\begin{aligned} \mathbf{p} &= (x_0 - x_s)\mathbf{I} + (y_0 - y_s)\mathbf{J} + (z_0 - z_s)\mathbf{K} \\ &= (1 - (-2))\mathbf{I} + (1 - 1)\mathbf{J} + (0 - 2)\mathbf{K} \\ &= 3\mathbf{I} - 2\mathbf{K} \end{aligned}$$

and compute

$$t = \frac{\mathbf{n} \cdot \mathbf{p}}{\mathbf{n} \cdot \mathbf{d}} = \frac{1 \times 3 + 1 \times 0 + 2 \times (-2)}{-1} = 1$$

The intersection point is at $(x_s + x_d, y_s + y_d, z_s + z_d) = (-1, 1, 1)$.

12.14 Double check that the intersection point found in Prob. 12.13 is indeed on the given plane.

SOLUTION

The vector equation for the plane is

$$x_n(x - x_0) + y_n(y - y_0) + z_n(z - z_0) = 0$$

Since

$$1(-1 - 1) + 1(1 - 1) + 2(1 - 0) = 0$$

the intersection point $(-1, 1, 1)$ is indeed on the plane.

12.15 Let S_1 be a sphere of radius 8 centered at $(2, 4, 1)$ and S_2 a sphere of radius 10 centered at $(10, -2, -5)$. Determine if a ray with $\mathbf{s} = 2\mathbf{J} + 5\mathbf{K}$ and $\mathbf{d} = \mathbf{I} - 2\mathbf{K}$ intersects the spheres.

SOLUTION

First consider sphere S_1. We have

$$\begin{aligned} A &= |\mathbf{d}|^2 = 1^2 + (-2)^2 = 5 \\ B &= (\mathbf{s} - \mathbf{c}) \cdot \mathbf{d} = [(0 - 2)\mathbf{I} + (2 - 4)\mathbf{J} + (5 - 1)\mathbf{K}] \cdot (\mathbf{I} - 2\mathbf{K}) = -10 \\ C &= |\mathbf{s} - \mathbf{c}|^2 - R^2 = (-2)^2 + (-2)^2 + 4^2 - 8^2 = -40 \end{aligned}$$

Since $B^2 - AC = (-10)^2 - 5(-40) = 300 > 0$, there are two t values:

$$t_1 = \frac{10 + \sqrt{300}}{5} = 2(1 + \sqrt{3}) > 0 \qquad \text{and} \qquad t_2 = \frac{10 - \sqrt{300}}{5} = 2(1 - \sqrt{3}) < 0$$

From these we know that the ray starts from inside S_1. Now considering sphere S_2, we have

$$\begin{aligned} A &= |\mathbf{d}|^2 = 1^2 + (-2)^2 = 5 \\ B &= (\mathbf{s} - \mathbf{c}) \cdot \mathbf{d} = [(0 - 10)\mathbf{I} + (2 + 2)\mathbf{J} + (5 + 5)\mathbf{K}] \cdot (\mathbf{I} - 2\mathbf{K}) = -30 \\ C &= |\mathbf{s} - \mathbf{c}|^2 - R^2 = (-10)^2 + (4)^2 + 10^2 - 10^2 = 116 \end{aligned}$$

Since $B^2 - AC = (-30)^2 - 5 \times 116 = 320 > 0$, there are also two t values:

$$t_3 = \frac{30 + \sqrt{320}}{5} = 6 + 1.6\sqrt{5} > 0 \qquad \text{and} \qquad t_4 = \frac{30 - \sqrt{320}}{5} = 6 - 1.6\sqrt{5} > 0$$

From these we know that the ray intersects S_2 in two different places. By comparing $t_1 \approx 5.464$, $t_3 \approx 9.578$, and $t_4 \approx 2.422$, we see that the ray first enters S_2, then leaves S_1, and finally exits from S_2 (the two spheres overlap each other).

12.16 The (implicit) equation for a sphere of radius R centered at the origin is $x^2 + y^2 + z^2 - R^2 = 0$. Determine if a ray $\mathbf{s} + t\mathbf{d}$ intersects the sphere.

SOLUTION

Let $\mathbf{s} = x_s\mathbf{I} + y_s\mathbf{J} + z_s\mathbf{K}$ and $\mathbf{d} = x_d\mathbf{I} + y_d\mathbf{J} + z_d\mathbf{K}$. Substitute $x_s + tx_d$, $y_s + ty_d$, and $z_s + tz_d$ for x, y, and z, respectively:

$$(x_s + tx_d)^2 + (y_s + ty_d)^2 + (z_s + tz_d)^2 - R^2 = 0$$

Expand and regroup terms:

$$(x_d^2 + y_d^2 + z_d^2)t^2 + 2(x_s x_d + y_s y_d + z_s z_d)t + (x_s^2 + y_s^2 + z_s^2) - R^2 = 0$$

or

$$|\mathbf{d}|^2 t^2 + 2\mathbf{s} \cdot \mathbf{d}t + |\mathbf{s}|^2 - R^2 = 0$$

This matches the quadratic equation derived in the text (see Sect. 12.4 under "Sphere"), with the center of the sphere being set to $(0, 0, 0)$.

12.17 The (implicit) canonical equation for an elliptic paraboloid is $x^2 + y^2 - z = 0$ (see Chap. 9, Sect. 9.10, Fig. 9-19). Determine if a ray $\mathbf{s} + t\mathbf{d}$ intersects the paraboloid.

SOLUTION

Let $\mathbf{s} = x_s\mathbf{I} + y_s\mathbf{J} + z_s\mathbf{K}$ and $\mathbf{d} = x_d\mathbf{I} + y_d\mathbf{J} + z_d\mathbf{K}$. Substitute $x_s + tx_d$, $y_s + ty_d$, and $z_s + tz_d$ for x, y, and z, respectively:

$$(x_s + tx_d)^2 + (y_s + ty_d)^2 - (z_s + tz_d) = 0$$

Expand and regroup terms:

$$(x_d^2 + y_d^2)t^2 + (2x_s x_d + 2y_s y_d - z_d)t + (x_s^2 + y_s^2 - z_s) = 0$$

When the ray is parallel to the z axis (i.e., $x_d = 0$ and $y_d = 0$), the equation degenerates into

$$-z_d t + x_s^2 + y_s^2 - z_s = 0$$

From this we can find

$$t = \frac{x_s^2 + y_s^2 - z_s}{z_d}$$

where

$$t \begin{cases} < 0 & \text{ray's negative extension intersects paraboloid} \\ = 0 & \text{ray's starting point on paraboloid} \\ > 0 & \text{ray intersects paraboloid} \end{cases}$$

Otherwise, we can rewrite the quadratic equation as

$$At^2 + Bt + C = 0$$

where

$$A = x_d^2 + y_d^2 \qquad B = 2x_s x_d + 2y_s y_d - z_d \qquad C = x_s^2 + y_s^2 - z_s$$

and the solution for the equation is (note $A \neq 0$)

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

If $B^2 - 4AC < 0$, the ray does not intersect (or touches) the paraboloid. If $B^2 - 4AC = 0$ and $t > 0$, the ray intersects (or touches) the paraboloid once. If $B^2 - 4AC > 0$, we get two t values; t_1 and t_2. If $t_1 < 0$ and $t_2 < 0$, the negative extension of the ray intersects the paraboloid (no intersection by the ray). If one of the two values is 0, the ray starts from a point on the paraboloid and intersects the paraboloid only if the other value is

positive. If t_1 and t_2 differ in signs, the ray originates from inside the paraboloid (i.e., $x_s^2 + y_s^2 - z_s < 0$) and intersects the paraboloid once. If both values are positive, the ray intersects the paraboloid twice (first enters and then exits), and the smaller value corresponds to the intersection point that is closer to the starting point of the ray.

12.18 Figure 12-19 shows a ray $\mathbf{s} + t\mathbf{d}$ and a canonical paraboloid (see Prob. 12.17) that is defined in its own coordinate system $x'y'z'$. Describe how to make use of the result in Prob. 12.17 to determine if the ray intersects this paraboloid.

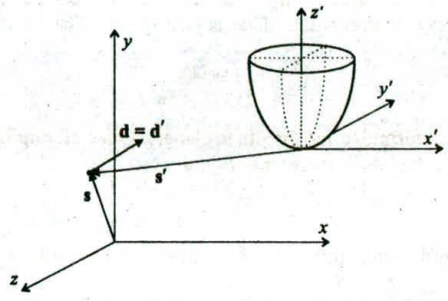


Fig. 12-19

SOLUTION

Find the coordinate transformation M from system xyz to $x'y'z'$. Apply M to transform (x_s, y_s, z_s) to (x'_s, y'_s, z'_s) and define $\mathbf{s}' = x'_s\mathbf{I} + y'_s\mathbf{J} + z'_s\mathbf{K}$. Apply M to transform the origin $(0, 0, 0)$ of xyz to (x'_0, y'_0, z'_0), and transform (x_d, y_d, z_d) to (x'_d, y'_d, z'_d). Define $\mathbf{d}' = (x'_d - x'_0)\mathbf{I} + (y'_d - y'_0)\mathbf{J} + (z'_d - z'_0)\mathbf{K}$, which describes the same vector as $\mathbf{d}$. Now represent the ray by $\mathbf{s}' + t\mathbf{d}'$ in the $x'y'z'$ coordinate system. Use the solution to Prob. 12.17 to determine if the ray intersects the paraboloid.

12.19 Refer to Fig. 12-12. If $k_{r1} = k_{r2} = k_{r3} = 0.3$, the threshold is 0.01, and the desired depth is 2, will the ray-tracer proceed to work on the third reflected ray?

SOLUTION

No, since the desired depth is 2. It would if the desired depth were 3 because $k_{r1}k_{r2}k_{r3} = 0.027 > 0.01$.

12.20 Describe a scene where the bounding volume techniques are definitely not applicable. Explain why.

SOLUTION

A number of spheres scattered across the scene. First of all, there is no simpler bounding volume for intersection testing. Furthermore, any bounding volume that encompasses several spheres is going to occupy a relatively large section of the scene. This brings about a relatively high probability for a ray to intersect the bounding volume, resulting in further testing against the enclosed spheres (something the bounding volume is supposed to help avoid in the first place).

12.21 In spatial subdivision we need to identify the regions along the ray's path. Since the ray exits from the current region and enters into the next region at the same point in space, a basic operation to guide the ray-tracer from one region to the next is: given the direction $\mathbf{d}$ of the ray and its entry point P_1 into the current region, find point P_2 through which the ray exits. Describe the operation for a parallelepiped region whose faces are parallel to the coordinate system planes.

SOLUTION

Such a region is bounded by the following planes:

$$x = x_{\min} \qquad x = x_{\max}$$
$$y = y_{\min} \qquad y = y_{\max}$$
$$z = z_{\min} \qquad z = z_{\max}$$

Since we are looking for the exit point, only planes that satisfy $\mathbf{n} \cdot \mathbf{d} > 0$, where $\mathbf{n}$ is the plane's normal vector, should be considered. The normal vectors of the planes listed above are simply $-\mathbf{I}$, $\mathbf{I}$, $-\mathbf{J}$, $\mathbf{J}$, $-\mathbf{K}$, and $\mathbf{K}$, respectively. Now we use P_1 to be the starting point of the ray. For each chosen plane, say, $z = z_{\min}$, the t value that represents where the ray intersects the plane is (see Prob. 12.8)

$$t = \frac{z_{\min} - z_s}{z_d}$$

(see Prob. 12.22 for similar formulae for the other planes). Once all eligible planes have been processed, the smallest t value from the calculation represents P_2.

12.22 Refer to Prob. 12.21, and write down the formulae for computing t for the other five planes.

SOLUTION

The formulae for planes $x = x_{\min}$, $x = x_{\max}$, $y = y_{\min}$, $y = y_{\max}$, and $z = z_{\max}$ are

$$t = \frac{x_{\min} - x_s}{x_d}, \qquad t = \frac{x_{\max} - x_s}{x_d}, \qquad t = \frac{y_{\min} - y_s}{y_d}, \qquad t = \frac{y_{\max} - y_s}{y_d}, \qquad t = \frac{z_{\max} - z_s}{z_d}$$

respectively.

12.23 Consider supersampling and adaptive supersampling. Which one is likely to perform better in reducing dropout?

SOLUTION

Supersampling is likely to perform better since the adaptive method will not subdivide a pixel if the center ray and corner rays return similar colors (but a small object may lie between those rays).

12.24 How to displace a ray from its normal position in the pixel grid?

SOLUTION

Let (x, y) be the ray's normal position in the grid. Define Δx and Δy such that $[x - \Delta x, x + \Delta x]$ and $[y - \Delta y, y + \Delta y]$ encompass the area for the new position (normally the area of the corresponding subpixel). Choose two random numbers δ_x and δ_y in the range of $[-\Delta x, \Delta x]$ and $[-\Delta y, \Delta y]$, respectively. The new/displaced position for the ray is $(x + \delta_x, y + \delta_y)$.

12.25 Illustrate why the relative size of an object affects the quality of environment mapping.

SOLUTION

The environment map is obtained by projecting the scene (a pyramid) from viewpoint C [see Fig. 12-20(a)]. When a small object is placed at C, the mapped reflection approximates the result of ray-tracing with the actual pyramid [see Fig. 12-20(b)]. On the other hand, if a relatively large object is placed at C, we may see a falsely reflected pyramid on the object when the pyramid is really behind the object [see Fig. 12-20(c)].

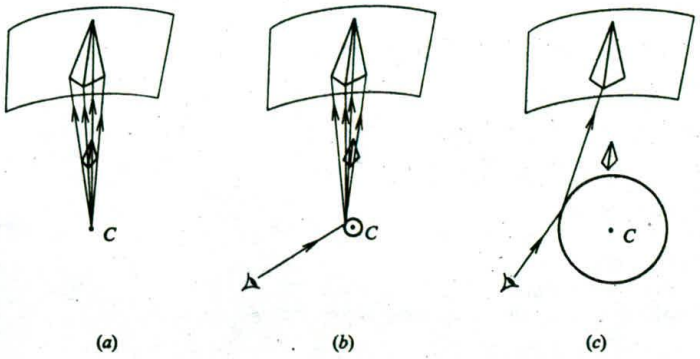


Fig. 12-20

12.26 Explain why the shape of an object affects the quality of environment mapping.

SOLUTION

A concave object may reflect part of itself. However, since the object is not depicted in the environment map, any self-reflection will be missing from the result of environment mapping.

Supplementary Problems

12.27 Snell's law is a consequence of Fermat's principle in optics: light travels from point A to point B along the path that takes the shortest time. Let A be in one medium where the speed of light is c_1, and B be in another medium where the speed of light is c_2 (see Fig. 12-21 where the x axis separates the two media). The path that takes the shortest time must be a straight line from A to a point P on the x axis followed by a straight line from P to B (since the light ray has to cross the x axis at some point and the shortest path within a medium is a straight line). Prove that the following holds for this path:

$$\frac{\sin(\alpha)}{\sin(\beta)} = \frac{c_1}{c_2}$$

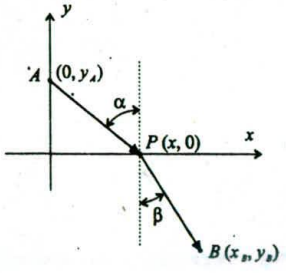


Fig. 12-21

12.28 The following pseudo-code procedure shows another way to describe the basic ray-tracing algorithm. The parameters are identical to those used in the procedure in Sect. 12.2, with the exception that the range for depth now includes value 0. Do these two procedures produce identical results when invoked with the same call? If yes, are they the same in terms of execution efficiency?

```
rayTrace(ray, depth, color)
{
  if (depth < 1) color = black;
  else {
     determine closest intersection of ray with an object;
     if (no intersection) color = background;
     else {
       color = local contribution;
       calculate reflected ray;
       rayTrace(reflected ray, depth-1, ref_color);
       calculate transmitted ray;
       rayTrace (transmitted ray, depth-1, trans_color);
       color = color + k_r · ref_color + k_t · trans_color;
     }
  }
}
```

12.29 Refer to Prob. 12.7, and convert $y = 2x - 6$ to parametric vector representation.

12.30 Determine if a ray intersects the yz plane.

12.31 The implicit equation for a cylinder of radius R along the z-axis is $x^2 + y^2 - R^2 = 0$. Determine if a ray $\mathbf{s} + t\mathbf{d}$ intersects the cylinder.

12.32 Show that the solution to equation $At^2 + 2Bt + C = 0$:

$$t = \frac{-B \pm \sqrt{B^2 - AC}}{A}$$

and the solution to equation $At^2 + Bt + C = 0$

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

are essentially the same.