CHAPTER TEN

LIGHT, COLOR, AND SHADING

INTRODUCTION

In previous chapters, we have seen how to construct three-dimensional objects out of polygons. We can now project and display perspective images of objects to get a feeling of depth. We have also learned how to remove hidden lines and surfaces to give the images a greater degree of realism. The polygons from which the objects are created can be filled in. Often, we will be able to draw polygons with different interior styles, and different styles may give the effect of different intensities or shades for the polygon surfaces. In this chapter we shall consider the shading of three-dimensional objects. We shall learn how to automatically set the polygon interior styles to give further realism to the image. We shall develop a model for the manner in which light sources illuminate objects, and use this model to determine how bright each polygon face should be. We shall also discuss how colors are described and how the model may be extended to colored objects.

DIFFUSE ILLUMINATION

Let us begin our discussion of illumination by considering an indirect lighting source. We will assume that our object is illuminated by light which does not come from any particular source but which comes from all directions. This is background light, light which is reflected from walls, floor, and ceiling. We assume that this light is uniform, that there is the same amount everywhere. There will be no bright spots, and no particular direction will be favored. We will assume that there is as much light going up as there is going down and that there is the same amount going right as there is going left. A surface will therefore receive the same amount of *diffuse* light energy no matter what its orientation may be.

Now, some of this energy will be absorbed by the surface, while the rest will be reflected or reemitted. The ratio of the light reflected from the surface to the total incoming light is called the *coefficient of reflection* or the *reflectivity*. A white surface reflects or reemits nearly all of the incoming radiation; hence its coefficient of reflection is close to 1. A black surface, on the other hand, absorbs most of the incoming light; hence its reflectivity is near zero. A gray surface would have a reflection coefficient somewhere in between. Most objects will reflect some colors of light better than other colors. Suppose that we have an object which has a high coefficient of reflection for red light and a low value for the other colors. If we illuminate the object with white light (which has equal amounts of all colors), then the red portion of the light will be reflected from the object while the other colors will be absorbed. The object will look red. In fact, this is the mechanism which gives objects their colors. If we are using a color graphics display which gives each pixel a red, green, and blue intensity value, we would have to specify three reflectivity values (one for each of the three colors) in order to know the color and shading properties of the object. But to keep things simple, we will begin by considering gray-level displays, and a single reflection coefficient number (R) will tell us what shade of gray our object is painted.

We said that the amount of light striking the object will not change with the object's position because the light is coming uniformly from all directions, but will the intensity of the light reflected to our eye change with different orientations? The answer is that the object will look equally bright no matter how it is positioned. This result arises from the cancellation of two effects. The first effect is that more light is emitted perpendicular to the surface than parallel to it. The precise relation is called *Lambert's law*, and it states that the reflection of light from a perfectly diffusing surface varies as the cosine of the angle between the normal to the surface and the direction of the reflected ray. (See Figure 10-1.)

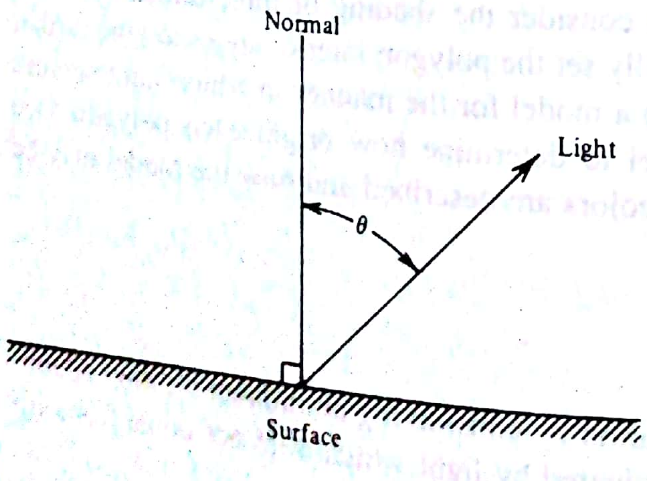

FIGURE 10-1
The direction of light is measured from the surface normal.

Thus the amount of light coming from a point on the surface would decrease as the cosine of the angle as that surface is rotated away from our viewpoint. The compensating effect arises from the fact that we don't see points but areas.

As the surface is turned away from our eyes, the number of light-emitting points within an area of view increases, and this increase happens to be by the reciprocal of the cosine. Therefore, as the surface is turned away from our eyes, we get less light from each point; but the points appear to be squeezed closer together, and the net effect is that the brightness of the surface is unchanged. (See Figure 10-2.)

A similar cancellation of effects occurs as the surface is moved farther from the viewpoint. Light coming from the surface is then spread out over a larger area. The area increases by the square of the distance, so the amount of the light reaching the eye is decreased by this same factor. By the same arguments, however, the objects being viewed will appear to be smaller, again by the same factor. So, while we have less light, it is applied to a smaller area on the retina and the brightness of the surface remains unchanged. (See Figure 10-3.)

We can now give an expression for the brightness of an object illuminated by diffuse background light. If B is the intensity of the background light and R is the object's reflectivity, then the intensity of light coming from any visible surface will be

$$v = BR \tag{10.1}$$

If we allow the user to set the values of B and R, he can create light or dark scenes and "paint" his objects different shades of gray. But in this simple model, every plane on a particular object will be shaded equally. This is not, however, the way real objects

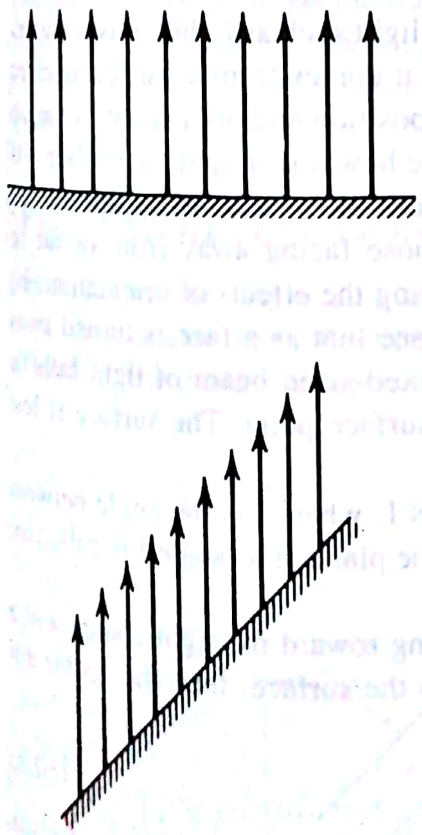

FIGURE 10-2
Viewing the surface at an angle increases the light-producing area in view.

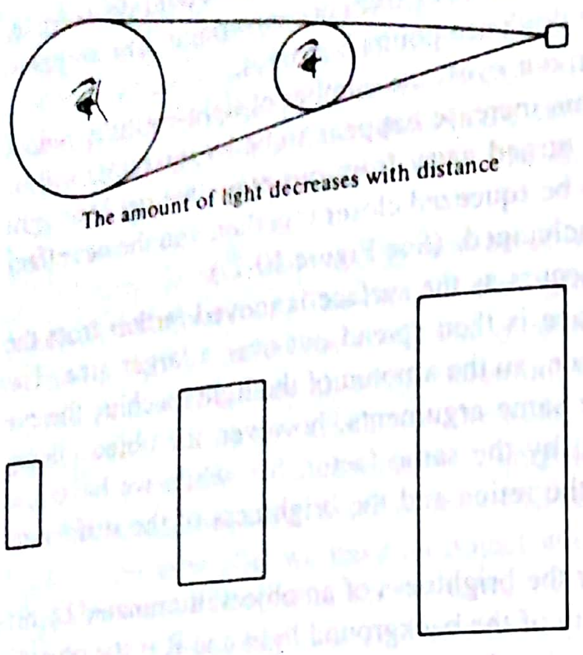
The amount of light decreases with distance
The apparent size of objects decreases with distance

FIGURE 10-3
The canceling effects for changes of distance.

look; and for a more realistic shading model, we must also include point sources of illumination.

POINT-SOURCE ILLUMINATION

Point sources are abstractions of real-world sources of light such as light bulbs, candles, or the sun. The light originates at a particular place, it comes from a particular direction over a particular distance. For point sources, the position and orientation of the object's surface relative to the light source will determine how much light the surface will receive and, in turn, how bright it will appear. Surfaces facing toward and positioned near the light source will receive more light than those facing away from or far removed from the illumination. Let's begin by considering the effects of orientation. By arguments similar to those we made earlier, we can see that as a face is turned away from the light source, the surface area on which a fixed-sized beam of light falls increases. This means that there is less light for each surface point. The surface is less brightly illuminated. (See Figure 10-4.)

The illumination is decreased by a factor of cos I, where I is the angle between the direction of the light and the direction normal to the plane. The angle I is called the *angle of incidence*. (See Figure 10-5.)

If we have a vector of length 1 called L pointing toward the light source and a vector of length 1 called N in the direction normal to the surface, then the vector dot product gives

$$\cos I = L \cdot N \tag{10.2}$$

Suppose that P amount of light comes from the point source. Then the shade of a surface of an object will be given by

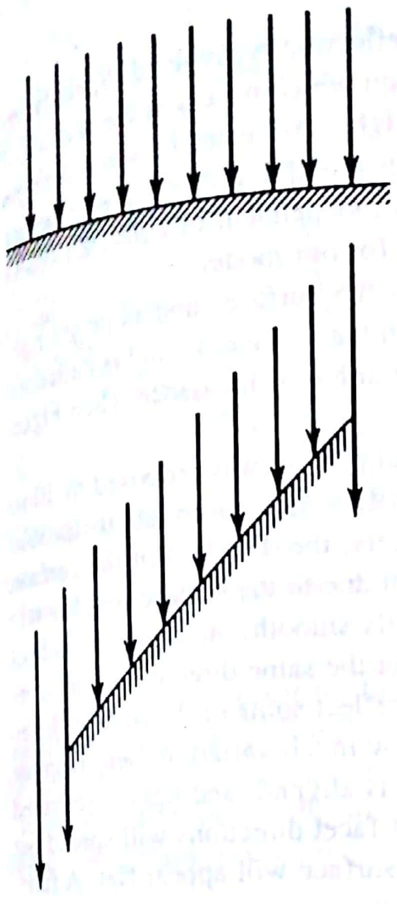

FIGURE 10-4
A surface facing the light source will receive more light than one at an angle.

$$v = BR + PR(L \cdot N) \quad (10.3)$$

The point-source term has the light from the point source, times the cosine of the angle of incidence (giving us how much light illuminates each point on the surface), multiplied by the coefficient of reflection (giving us how much of this light is reemitted to reach the viewer).

SPECULAR REFLECTION

Our model can be improved even more. Light can be reflected from an object in two ways: there is a diffuse reflection, which we have included above, and there is *specular* reflection. The diffuse reflection depends only upon the angle of incidence, and the re-

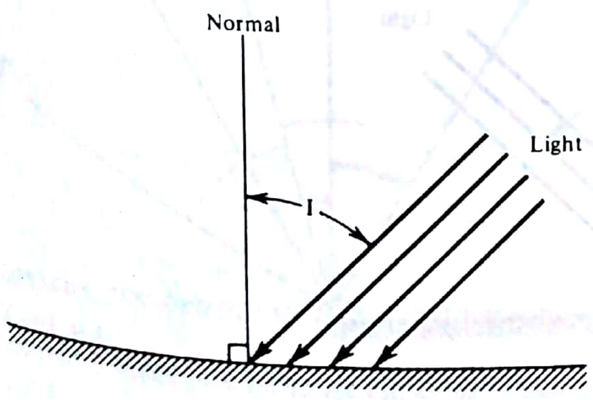

FIGURE 10-5
The angle of incidence.

flected light can be colored, since the coefficient of reflection is involved. Specular reflection acts quite differently. It is the type of reflection which occurs at the surface of a mirror. (Note the reflected light in Plates 5 and 6.) Light is reflected in nearly a single direction (not spread out over all directions in accordance with Lambert's law). Plastics and many metals have a specular reflection which is independent of color; all colors are equally reflected. We shall assume this is the case for our model.

For specular reflection, light comes in, strikes the surface, and bounces right back off. The angle that the reflected beam makes with the surface normal is called the *angle of reflection* O, and is equal in magnitude to the angle of incidence. (See Figure 10-6.)

The model for specular reflection which we shall present was proposed by Blinn [BLI77]. According to this model, there are four factors which contribute to specular reflection. They are the distribution of the surface facets, the direction of the surface, the absorption of the surface, and the blockage of light due to the surface roughness. The model assumes that the surface is not perfectly smooth, but rather is made of tiny mirrorlike *facets*. Most of the facets will face about the same direction as the overall surface, but some will be a little different and will reflect some of the light in different directions. How glossy an object is depends on how much variation there is in the surface facets. A very glossy object has almost all facets aligned, and light is reflected in a sharp ray. An object with a broader distribution of facet directions will spread out the light; reflections will blur and merge; the object's surface will appear flat. A function which describes the distribution of facets is given by

$$D = \left[\frac{k}{k + 1 - (N \cdot H)}\right]^2 \tag{10.4}$$

The vector H is a unit vector in the direction halfway between the light and the eye. The light from the specular reflection can be seen when the direction from the object to the eye is the same direction as that of the reflected light. Another way to state this is to form a vector halfway between the direction of the incident light and the direction to the eye. (See Figure 10-7.)

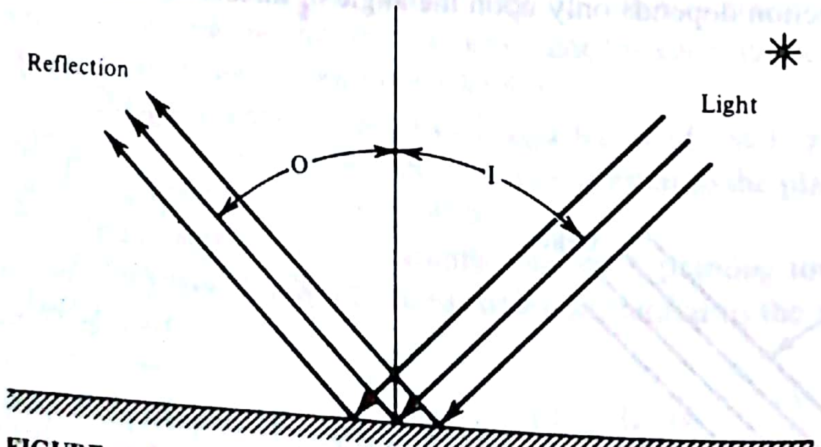

FIGURE 10-6
Specular reflection.

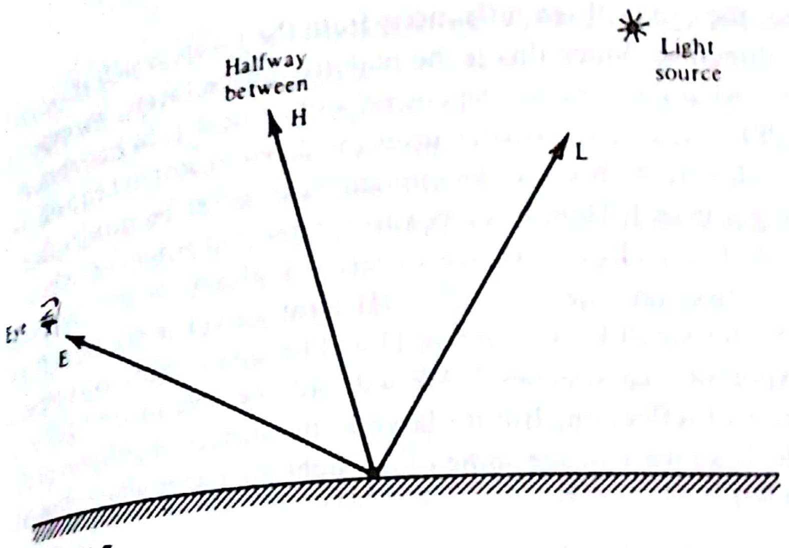

FIGURE 10-7
Defining the vector halfway between the light source and the viewer's eye.

Compare this vector to the surface normal, which is halfway between the incident and reflected rays. (See Figure 10-8.) If the directions match, then the eye can see the reflected rays. Let's first see how to compute the vector H. If L is a vector of length 1 in the direction out of the surface along the incident light ray, and E is a vector of length 1 in the direction of the eye, then

$$H = \frac{L + E}{|L + E|} \tag{10.5}$$

will be a vector of length 1 pointing halfway between them. Now we want to show that Equation 10.4 gives the desired behavior for light reflected from a surface of facets, most of which are aligned with the surface normal. If N is a vector of length 1 in the direction of the surface normal, then the dot product N · H will give the cosine of the angle between the halfway vector and the surface normal. When the angle between

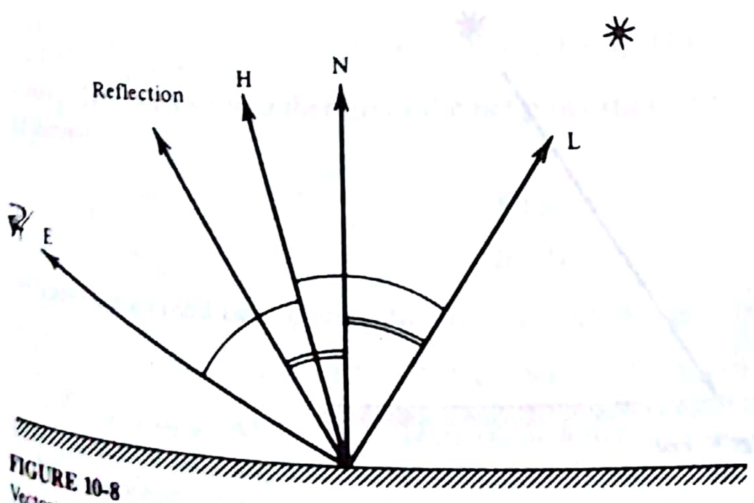

FIGURE 10-8
Vectors used to determine when the eye can see the reflection.

these two vectors is zero, the eye will see reflections from the facets which are aligned with the overall surface direction. Since this is the majority of the facets, we expect D to be largest for this case. At angles near 0, the cosine will be near 1. In Equation 10.4 the term 1 − (N · H) will be close to 0, which causes the denominator to be small, and D is its strongest for this direction. But the denominator can never be smaller than k, and D cannot become larger than 1. Here k is a positive parameter which describes the glossiness of the material. For k close to 0, the material is glossy; as the eye moves away from the angle of reflection, the 1 − (N · H) term moves away from 0 and quickly overwhelms k. So for small k, we see that D will be small unless N is close to H, at which point the expression approaches 1. We will only be able to see light if we are looking along the angle of reflection. But for large k, the surface is uniform or flat. D will always be close to 1, so we will see some of the light no matter which direction we look. (See Figure 10-9.)

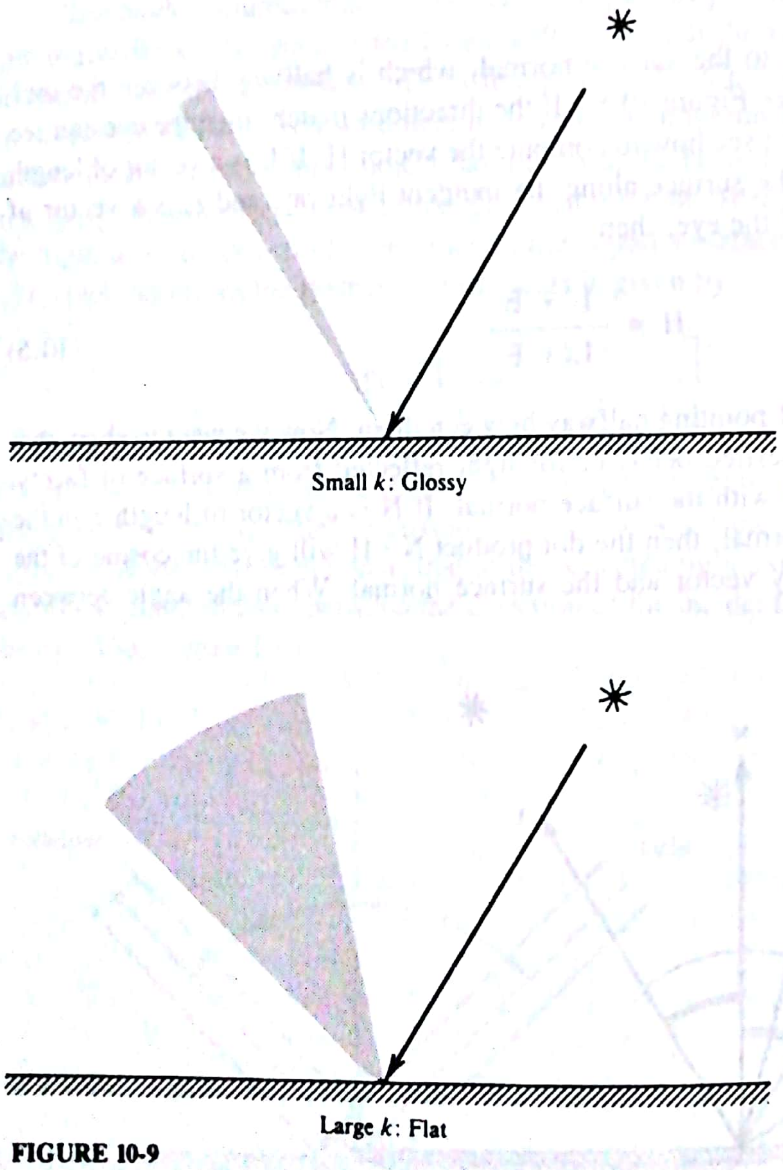

FIGURE 10-9
The effect of k.

A second factor is how much of the lighted surface is turned away from our eyes. As we have seen, when a surface is turned away from our eyes, the number of light-emitting points increases by a factor of the reciprocal of the cosine of the angle [i.e., 1 / (N · E)]. For the diffuse case, this effect was canceled by Lambert's law, but for specular reflection we should retain it.

The third factor is that not all of the light is reflected; some of it is absorbed (or transmitted if the object is transparent). This depends upon the angle of view. The theoretical expression for the amount of light reflection for nonabsorbing materials is given by the *Fresnel equation*

$$F = \frac{(g - c)^2}{2(g + c)^2} \left\{ 1 + \frac{[c(g + c) - 1]^2}{[c(g - c) + 1]^2} \right\} \tag{10.6}$$

where

$$c = E \cdot H \tag{10.7}$$

is the cosine of half the angle between the eye and the light, and g is given by

$$g = (n^2 + c^2 - 1)^{1/2} \tag{10.8}$$

Here n is the effective index of refraction for the material. Transparent objects have an index of refraction near 1, giving small F values; whereas metals have a high value of n which results in F near 1, meaning that most of the light is reflected.

Finally, the roughness of the surface may block some of the light. (See Figure 10-10.) An expression for the amount of reflected light blocked by the surface roughness is

$$G_r = 2 (N \cdot H) (N \cdot E) / (E \cdot H) \tag{10.9}$$

The incident light can also be blocked by the surface roughness

$$G_i = 2 (N \cdot H) (N \cdot L) / (E \cdot H) \tag{10.10}$$

The light blockage factor G which should be used is

$$G = \min[1, \min(G_r, G_i)] \tag{10.11}$$

Putting these factors together gives the net proportion of the light available for specular reflection.

$$\frac{DGF}{E \cdot N} \tag{10.12}$$

We can now extend our shading function to include specular reflection.

$$v = BR + PR(L \cdot N) + PDGF / (E \cdot N) \tag{10.13}$$

Finally, let's include the effects of distance. In theory, the illumination from a point source should decrease as the square of the distance between it and the object being illuminated. Using this relation, however, seems to give too much change with

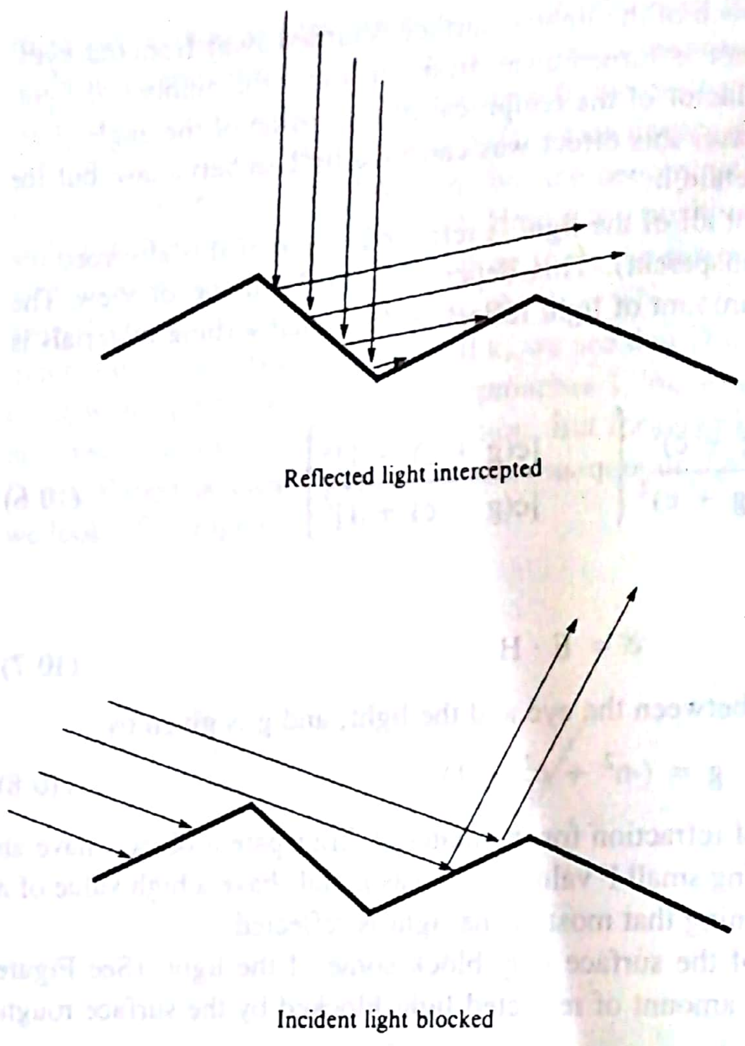

FIGURE 10-10
The effects of surface roughness.

distance. Perhaps this is partly due to the fact that most light sources are much larger than points. An object illuminated by a very large nearby light source would hardly show any change in illumination with small changes in distance. In any case, we want some function which decreases with distance less rapidly than the square. And we may also wish it to approach some finite limit for very small distances, so that we need not worry about division by zero for a badly positioned light source. We suggest the following function

$$W = \frac{1}{C + U} \tag{10.14}$$

where C is a positive constant and U is the distance from the light source to the object. Our shading expression then becomes

$$v = BR + \frac{P[R(L \cdot N) + DGF/(E \cdot N)]}{C + U} \tag{10.15}$$

We could generalize this to several point light sources if we wished.

$$v = BR + \sum_j \frac{P_j[R(L \cdot N) + D_jG_jF/(E \cdot N)]}{C + U_j} \tag{10.16}$$

SHADING ALGORITHMS

Now let's consider how to introduce automatic shading into our graphics system. We begin by creating a means to turn the automatic shading on or off. The flag SHADE-FLAG can be set by the user to indicate whether or not automatic shading should occur.

10.1 Algorithm SET-SHADING(ON-OFF) User routine to indicate automatic shading

```
Argument   ON-OFF the user's shading setting
Global     SHADE-FLAG a flag to indicate automatic shading
BEGIN
   SHADE-FLAG ← ON-OFF;
   RETURN;
END;
```

The shading formulas which we presented make use of the dot product of vectors of length 1 for the determination of cosines. We shall therefore often be normalizing vectors (scaling to length 1). It will be convenient to have a utility procedure to do this. The following routine normalizes a vector by dividing by its length. The vector's original length is also returned.

10.2 Algorithm UNITY(DX, DY, DZ, LENGTH) Utility routine to normalize a vector and compute its length

```
Arguments  DX, DY, DZ the vector to be normalized
           LENGTH for return of the vector's original length
BEGIN
   LENGTH ← SQRT(DX ↑ 2 + DY ↑ 2 + DZ ↑ 2);
   DX ← DX / LENGTH;
   DY ← DY / LENGTH;
   DZ ← DZ / LENGTH;
   RETURN;
END;
```

We must determine the shade of each polygon, using vectors and angles between it and the light sources and viewpoint. These vectors and angles must be determined before projection (which alters them). We shall alter the DISPLAY-FILE-ENTER routine to include setting of the CURRENT-FILL-STYLE to the proper shade.

10.3 Algorithm DISPLAY-FILE-ENTER(OP) (Revision of algorithm 8.38) Routine to enter an instruction into the display file

```
Argument   OP opcode of instruction to be entered
Global     DF-PEN-X, DF-PEN-Y, DF-PEN-Z the current pen position
           PERSPECTIVE-FLAG perspective vs. parallel projection flag
Local      X, Y, Z hold the point that is transformed
BEGIN
   IF OP < 1 AND OP > −32 THEN PUT-POINT(OP, 0, 0)
   ELSE
      BEGIN
         X ← DF-PEN-X;
```

```
            Y ← DF-PEN-Y;
            Z ← DF-PEN-Z;
            VIEW-PLANE-TRANSFORM(X, Y, Z);
            IF SHADE-FLAG THEN SHADE-CHECK(OP, X, Y, Z);
            IF PERSPECTIVE-FLAG THEN PERSPECTIVE-TRANSFORM(X, Y, Z)
            ELSE PARALLEL-TRANSFORM(X, Y, Z);
            CLIP(OP, X, Y, Z);
        END;
    RETURN;
END;
```

We have called SHADE-CHECK in order to calculate the appropriate shade for polygons. The statements in SHADE-CHECK determine the polygon's normal vector and its center. The routine then calls MAKE-SHADE to actually find the shading factor. The SHADE-CHECK procedure is called for every vertex of the polygon, but will calculate the shade only on the last vertex call. The first call will be for the polygon instruction. This causes initializations of the variables and sets a counter for the number of vertices. For each of the following vertices, we accumulate the average position and calculate the normal vector as we did for back-face removal. On the last vertex of the polygon, we complete the center position calculation by averaging the accumulated vertex coordinates. We then use MAKE-SHADE to determine the appropriate interior style. Note that the method we use to find the normal vector for the polygon will give reasonable results when the vertex coordinates are close to but not exactly planar. This can be useful in finding the shade of small curved surface patches as are discussed in Chapter 11.

10.4 Algorithm SHADE-CHECK(OP, X, Y, Z) Calculates the shade of polygons Determines the normal and center; then calls MAKE-SHADE

```
Arguments   OP, X, Y, Z a display-file instruction
Global      A-SHADE, B-SHADE, C-SHADE for accumulating the polygon normal
            X-SHADE, Y-SHADE, Z-SHADE for accumulating the polygon center
            X-PREV, Y-PREV, Z-PREV the previous vertex
            SIZE-SHADE the polygon size
            SIDE-COUNT a counter of polygon sides seen
BEGIN
    IF OP > 2 THEN
        BEGIN
            new polygon instruction
            SIZE-SHADE ← OP;
            A-SHADE ← 0;
            B-SHADE ← 0;
            C-SHADE ← 0;
            X-SHADE ← 0;
            Y-SHADE ← 0;
            Z-SHADE ← 0;
            X-PREV ← X;
            Y-PREV ← Y;
            Z-PREV ← Z;
        SIDE-COUNT ← 0;
        RETURN;
    END;
IF SIZE-SHADE > 0 THEN
    BEGIN
        enter another side
        A-SHADE ← A-SHADE + (Y-PREV − Y) * (Z-PREV + Z);
        B-SHADE ← B-SHADE + (Z-PREV − Z) * (X-PREV + X);
        C-SHADE ← C-SHADE + (X-PREV − X) * (Y-PREV + Y);
        X-SHADE ← X-SHADE + X;
        Y-SHADE ← Y-SHADE + Y;
        Z-SHADE ← Z-SHADE + Z;
        X-PREV ← X;
        Y-PREV ← Y;
        Z-PREV ← Z;
        SIDE-COUNT ← SIDE-COUNT + 1;
        IF SIDE-COUNT = SIZE-SHADE THEN
            BEGIN
                all vertices seen, so average center position
                X-SHADE ← X-SHADE / SIZE-SHADE;
                Y-SHADE ← Y-SHADE / SIZE-SHADE;
                Z-SHADE ← Z-SHADE / SIZE-SHADE;
                now determine the shade
                MAKE-SHADE;
                SIZE-SHADE ← 0;
            END;
    END;
  RETURN;
END;
```

Now let's discuss the MAKE-SHADE shading routine. This routine obtains as global variables the polygon's normal vector and a center point. It also obtains the position of the light source and the surface properties of the object. The normal vector should be scaled to length 1 by the UNITY routine. The difference between the position of the light source and the position of the polygon gives a vector for the direction to the light. The calculation for the direction to the eye depends upon whether a parallel or perspective projection is used. For a parallel projection, we take the parallel projection vector. For a perspective projection, the direction to the eye is the difference between the center of projection and the average polygon position.

After normalizing the light- and eye-direction vectors, they can be added to calculate the vector midway between them. Now the cosines can be determined. The cosine of the angle of incidence is the dot product of the normal vector and the light vector. A negative result means that the light is behind the surface and the shade will result only from the background illumination. We can plug all the parameters into our shading formula to get a number characterizing the object's brightness. Here the background illumination B (named BKGRND in the algorithm) is assumed to have values between zero and 1, and the point source is given by

$$P = 1 - B \tag{10.17}$$

The result is then scaled by an overall brightness factor BRIGHT. After determining the brightness of a polygon, the result is passed to the routine INTENSITY, which sets the appropriate polygon interior style.

10.5 Algorithm MAKE-SHADE Routine to calculate and set the shading style for a polygon

Global	XL, YL, ZL the location of the light point source
	PERSPECTIVE-FLAG the perspective vs. parallel projection flag
	XC, YC, ZC the center of perspective projection
	VXP, VYP, VZP the direction of parallel projection
	BKGRND the proportion of the light in background illumination
	REFL the reflectivity of the polygon
	K-GLOSS the polygon shininess
	BRIGHT the overall illumination scale vector
	X-SHADE, Y-SHADE, Z-SHADE the position of the polygon
	A-SHADE, B-SHADE, C-SHADE the normal vector to the polygon
	NRF the index of refraction of the surface
Local	DXL, DYL, DZL the direction of the light source
	U the distance to the lignt source
	DXE, DYE, DZE the direction of the eye
	DXH, DYH, DZH vector halfway between eye and light
	COSNL the cosine of the angle of incidence
	COSNH the cosine of angle between normal and halfway vectors
	COSNE the cosine of the angle between normal and eye vectors
	COSEH the cosine of the angle between eye and halfway vectors
	G factor for self-shading due to surface roughness
	GF, T intermediate values
	F reflection factor from Fresnel's equation
	D facet distribution of the surface
	SHADE-SETTING resulting shading value
	DUMMY a dummy variable

```
BEGIN
    normalize the surface normal vector
    UNITY(A-SHADE, B-SHADE, C-SHADE, DUMMY);
    calculate other vectors
    DXL ← XL − X-SHADE;
    DYL ← YL − Y-SHADE;
    DZL ← ZL − Z-SHADE;
    UNITY( DXL, DYL, DZL, U)
    IF PERSPECTIVE-FLAG THEN
        BEGIN
            DXE ← XC − X-SHADE;
            DYE ← YC − Y-SHADE;
            DZE ← ZC − Z-SHADE;
        END
    ELSE
        BEGIN
            DXE ← VXP;
            DYE ← VYP;
        DZE ← VZP;
    END;
UNITY(DXE, DYE, DZE, DUMMY);
DXH ← DXE + DXL;
DYH ← DYE + DYL;
DZH ← DZE + DZL;
UNITY(DXH, DYH, DZH, DUMMY);
calculate the cosines
COSNE ← A-SHADE * DXE + B-SHADE * DYE + C-SHADE * DZE;
IF COSNE ≤ 0 THEN RETURN;
it's a back face
COSNL ← A-SHADE * DXL + B-SHADE * DYL + C-SHADE * DZL;
IF COSNL ≤ 0 THEN
    BEGIN
        does not face the light source
        SHADE-SETTING ← MIN(1, BRIGHT * BKGRND * REFL);
        SET-FILL-STYLE(INTENSITY(SHADE-SETTING));
        RETURN;
    END;
COSNH ← A-SHADE * DXH + B-SHADE * DYH + C-SHADE * DZH;
COSEH ← DXH * DXE + DYH * DYE + DZH * DZE;
surface roughness
IF COSNE < COSNL THEN G ← 2 * COSNE * COSNH / COSEH
ELSE G ← 2 * COSNL * COSNH / COSEH;
IF G > 1 THEN G ← 1;
reflection factor
GF ← SQRT(NRF * NRF + COSEH * COSEH − 1);
T ← ((GF + COSEH) * COSEH − 1) / ((GF − COSEH) * COSEH + 1);
F ← T * T + 1;
T ← (GF − COSEH) / (GF + COSEH);
F ← F * T * T / 2;
facet distribution
D ← K-GLOSS / (K-GLOSS + 1 − COSNH);
D ← D * D;
calculate the shading value
SHADE-SETTING ← MIN(1, (BKGRND * REFL + (1 − BKGRND) * (REFL *
    COSNL + G * F * D / COSNE) / (1 + U) ) * BRIGHT);
SET-FILL-STYLE( INTENSITY(SHADE-SETTING));
RETURN;
END;
```

The INTENSITY routine converts shading values into settings for the polygon interior style. The actual implementation of this routine can depend upon the display device being used. In particular, the number of intensity values actually available and their corresponding fill-style parameters can differ for different implementations. The range of SHADE-SETTING values passed to the INTENSITY routine is 0 to 1. If the display device has possible intensity settings of 1 through N, these values may be generated by

$$\mathrm{INT(SHADE\text{-}SETTING} * (N - 1) + 1)$$

This is a linear transformation which assumes that the intensity varies linearly between settings. This is not true for some display devices, and in such cases, different formulas which compensate for the nonlinear hardware may have to be used. This is discussed further in the section on gamma correction. If the system has been written such that fill style 1 is the darkest, style 2 is lighter, and so on up through style N, then the above formula will give the correct style settings. An example of a case where the intensities may not correspond simply to the interior style numbers is the case where output is done on a line printer and different characters are used to get the different intensities. When the relation of style number to intensity is complex, an array can be used for the conversion. We store the fill-style codes in the array. The code for the darkest style is saved in the first array cell, the next darkest in the second cell, and so on, so that the code for the lightest style is in the Nth cell. Then we use the above formula to determine which cell to examine, and we retrieve from that cell the correct style setting. To actually make the change of style, we put the new style instruction into the display file. An example of an INTENSITY routine which could be used with a line printer is given below.

```
10.6 Algorithm INTENSITY(SHADE-SETTING) Example of routine to enter correct
interior style for shading
Argument    SHADE-SETTING the polygon's shading value
Global      SHADE-STYLES an array of 16 interior styles for shading
Local       SHADE-INDEX index of the style setting for the shade
BEGIN
    SHADE-INDEX ← INT(15 * SHADE-SETTING + 1);
    INTENSITY ← SHADE-STYLE[SHADE-INDEX];
    RETURN;
END;
```

We have used a number of parameters to characterize the light source and the object being illuminated. Of course, we must provide the means for the user to set the values for these parameters. We shall therefore write routines for placing the user's specifications into global variables. Objects will be characterized by their reflectivity (REFL), their index of refraction (NFR), and their glossiness (K-GLOSS). The following routine sets the object parameters.

```
10.7 Algorithm SET-OBJECT-SHADE(REFLECTIVITY, REFRACTION,
GLOSS) User routine to set an object's shading parameters
Arguments   REFLECTIVITY object's coefficient of reflection
            REFRACTION the index of refraction
            GLOSS the narrowness of the reflected rays
Global      REFL, NRF, K-GLOSS global storage for the object's parameters
BEGIN
    REFL ← REFLECTIVITY;
    NRF ← REFRACTION;
    K-GLOSS ← GLOSS;
    RETURN;
END;
```

We have allowed the object parameters to be changed at any time, so that as the scene is constructed, different objects may be given different appearances. We wish, however, to treat the lighting parameters somewhat differently. The lighting arrangement will be considered part of the viewing specification, and like the other viewing parameters discussed in Chapter 8, the lighting parameters should be fixed throughout the display-file segment. To do this, we shall have two sets of parameters: one set which the user can alter at any time, and a second set which is given the current user's specification whenever a display-file segment is created. It is the second set which is actually used in the shading calculation. The following routine gets the user's specification.

10.8 Algorithm SET-LIGHT(X, Y, Z, BRIGHTNESS, BACKGROUND) User routine for saving the lighting parameters

```
Arguments   X, Y, Z the location of the light source
            BRIGHTNESS an overall-intensity scaling factor
            BACKGROUND the portion of the light which is indirect
Global      XL1, YL1, ZL1, BRT, BKG storage for the user's specification
BEGIN
    XL1 ← X;
    YL1 ← Y;
    ZL1 ← Z;
    BRT ← BRIGHTNESS;
    BKG ← MAX(0, MIN(1, BACKGROUND));
    RETURN;
END;
```

When a display-file segment is created, we shall copy the current lighting parameters into the global variables which will be used in the shading calculation. We shall also convert the light-source position to the view plane coordinate system. The COPY-LIGHT routine carries out these operations.

10.9 Algorithm COPY-LIGHT Routine to update the current lighting parameters

```
Global      XL1, YL1, ZL1 the current light-source position
            BRT the current brightness
            BKG the current background illumination
            XL, YL ZL lighting position for the display-file segment
            BRIGHT the brightness for the display-file segment
            BKGRND the background illumination for the display-file segment
BEGIN
    BRIGHT ← BRT;
    BKGRND ← BKG;
    XL ← XL1;
    YL ← YL1;
    ZL ← ZL1;
    VIEW-PLANE-TRANSFORM(XL, YL, ZL);
    RETURN;
END;
```

The COPY-LIGHT routine should be called as part of the establishment of the viewing parameters. We therefore modify the NEW-VIEW-3 routine as follows:

10.10 Algorithm NEW-VIEW-3 (Revision of algorithm 8.39) Create a new overall viewing transformation

```
BEGIN
    MAKE-VIEW-PLANE-TRANSFORMATION;
    MAKE-Z-CLIP-PLANES;
    NEW-VIEW-2;
    COPY-LIGHT;
    RETURN;
END;
```

To complete our shading algorithm, we must include initializations for the parameters which we have created.

10.11 Algorithm INITIALIZE-10 Initialization of the shading parameters

```
BEGIN
    SIZE-SHADE ← 0;
    SET-LIGHT(0, 0, 1, 1, 0);
    INITIALIZE-9;
    SET-OBJECT-SHADE(1, 10, 10)
    SET-SHADING(FALSE);
    RETURN;
END;
```

The light source is initialized to location (0, 0, 1) with brightness 1. Brightness values will normally be about 1 to 2, but may be larger to compensate for a distant light source or low reflectivity of an object. The background illumination is initialized to 0. This parameter should range between 0 (no indirect lighting) and 1 (totally indirect illumination). Objects are initialized as totally reflective but not very glossy.

SMOOTH SHADING OF SURFACE APPROXIMATIONS

So far we have dealt only with flat polygons. If we wanted to draw a curved shape (a doughnut, ball, can, cup, or airplane), we could approximate it with many small polygons. (See Figure 10-11.)

If we shade the approximate surface, we can often see the individual polygon edges. The edges are enhanced by the properties of our eyes to produce an effect called *Mach banding*. When a uniform dark surface meets a uniform light surface, the dark surface appears even darker along the edge, and the light surface even lighter. This makes the edge stand out. (See Figure 10-12.)

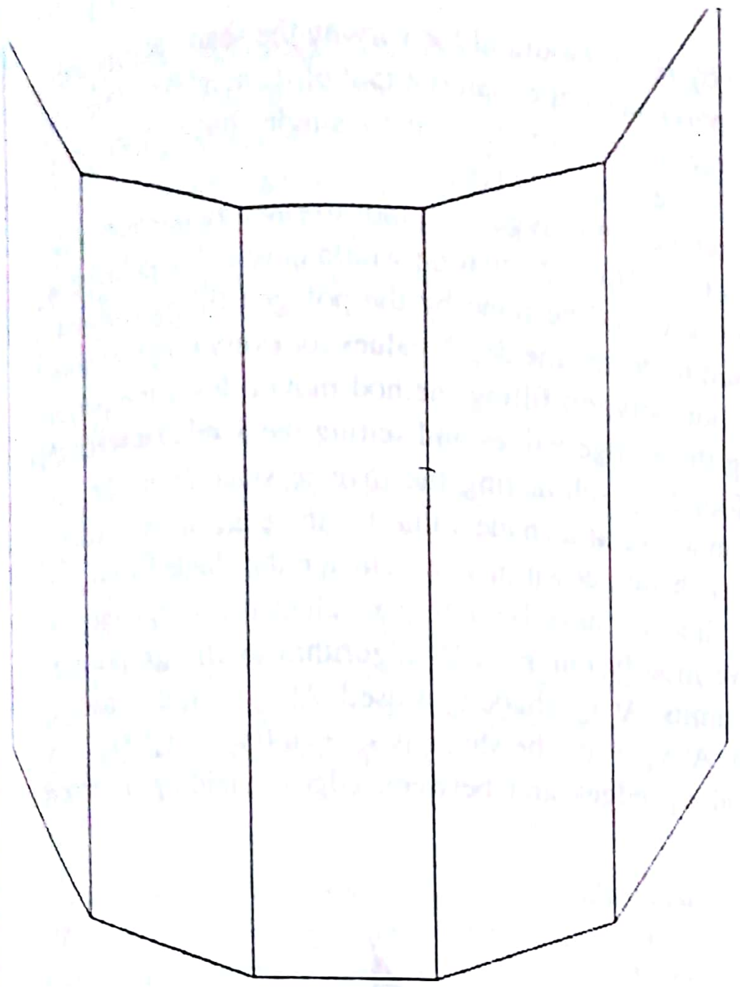

FIGURE 10-11
Approximating a curve with flat polygons.

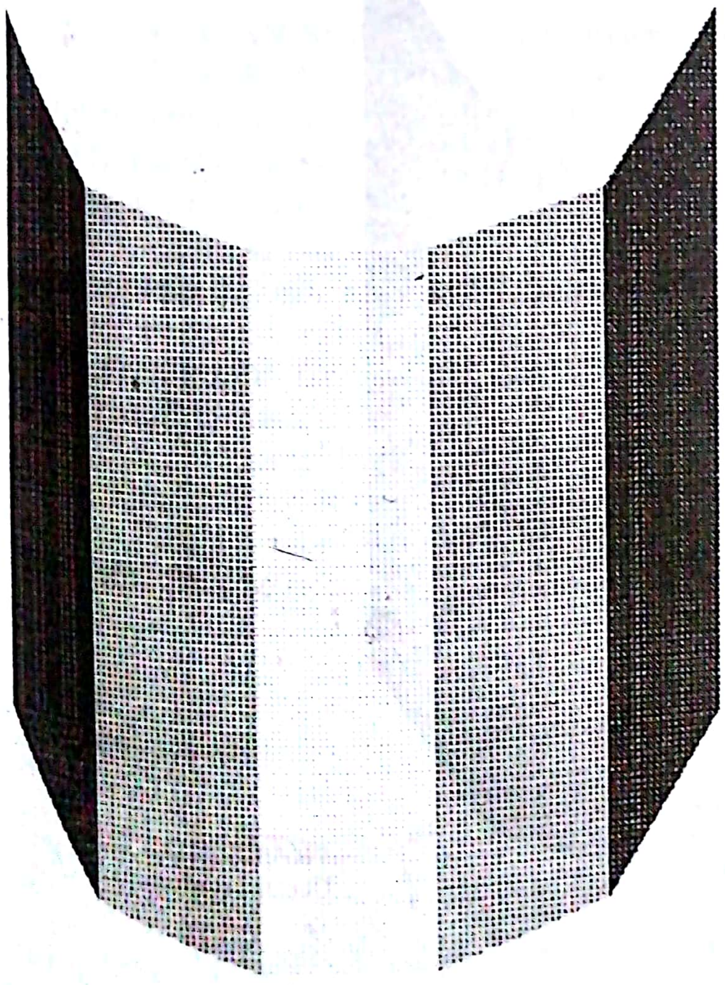

FIGURE 10-12
Flat shading makes the approximation noticeable.

A method was suggested by Henri Gouraud for varying the shade across polygon surfaces so that the shade at a polygon's edge matches that of its neighbor. (See Figure 10-13.) To do this, we need more information than just a single shade value per polygon. Instead, we need a shade value at each vertex.

We could, for example, extend our polygon routines to enter an intensity value at each vertex, along with the position. This information would have to be included in the display file. The rest of the work would be done by the polygon-filling routines. We use the same method that is used to determine depth values for every polygon point for use in a Z buffer. Recall that our polygon-filling method moves down the polygon a scan line at a time, calculating the x edge values and setting the pixels between them. The new x values could be found by subtracting the inverse slope from the current values. In a similar way we can arrive at a shade value for the edge at each scan line. We start with s_1 at y_1, and for each step down in y, we change the shade by $(s_2 - s_1) / (y_2 - y_1)$. Now for each scan line we have bounding x values (x_a to x_b) and bounding shade values (s_a to s_b). We modify our FILLIN algorithm to change pixel intensities smoothly between these limits. At x_a, shade s_a is used. At $x_a + 1$, the shade $s_a + [(s_b - s_a) / (x_b - x_a)]$ is used. At $x_a + n$, the shade is $s_a + n [(s_b - s_a) / (x_b - x_a)]$. So shading changes linearly along edges and between edges, yielding a smoother, more natural appearance.

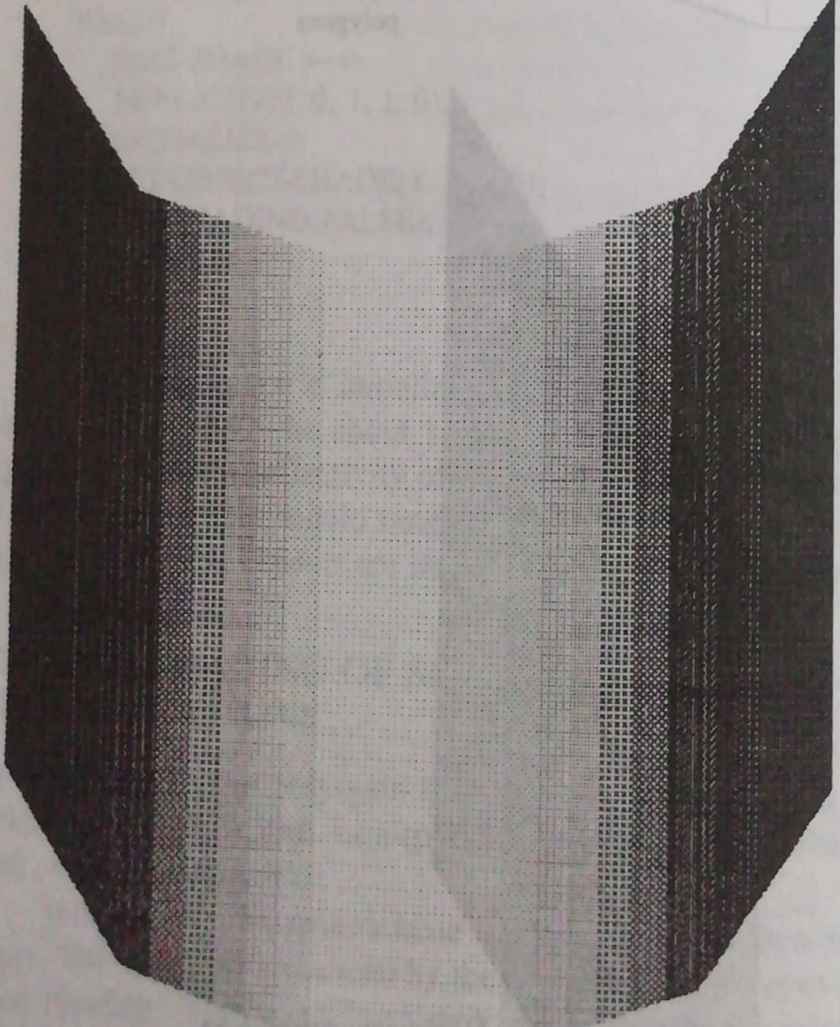

FIGURE 10-13
Gouraud shading.

Specifying the shade of each vertex in the polygon routine leaves it up to the user to determine the appropriate shading. It is also possible to perform automatic shading of curved surfaces using Gouraud's interpolation scheme. The problem is to automatically determine the shade of the surface at each vertex. The formulas that we have presented may be used, only the shade value at a vertex should be based on the direction of the true surface at that point and not the direction of the polygon approximation. (See Figure 10-14.)

TRANSPARENCY

Our shading model has not considered transparent objects. The surface of a transparent object may receive light from behind as well as from in front. Such objects have a *transparency coefficient* T as well as values for reflectivity and specular reflection. The transparency coefficient may depend upon the thickness of the object. If one layer of tinted glass lets through only half of the light, then two layers will let through only one quarter. The transmission of light depends exponentially on the distance which the light ray must travel within the object.

$$T = te^{-ad} \qquad (10.18)$$

In this expression, (d) is the distance the light must travel in the object. The coefficients (t) and (a) are properties of the material. The (t) determines how much of the light is transmitted at the surface (instead of reflected), and (a) tells how quickly the material absorbs or attenuates the light. (See Figure 10-15.)

If we view a light source through a transparent object, the properties of the surface (t) may be modeled in much the same way as for a reflecting surface. We are mainly interested in how much light passes through the surface. This is the light which is not reflected and is given by the Fresnel factor (1 − F). For most objects, the light both enters and exits, and a factor is needed for both surfaces.

The transparency and absorption coefficients can depend on color. Some objects let only red light through, while others attenuate the red and blue, letting only green pass. If we are dealing with colored objects, we need three pairs of transparency and absorption coefficients. For very clear objects, we can neglect the attenuation with distance or include it as an average value as part of (t).

A proper treatment of transparent objects should also consider the fact that light changes direction when it crosses the boundary between two media. (See Figure 10-16.) This effect is called *refraction*. The change in direction is related to a property of materials called the index of refraction (n). The exact relation is called *Snell's law.*

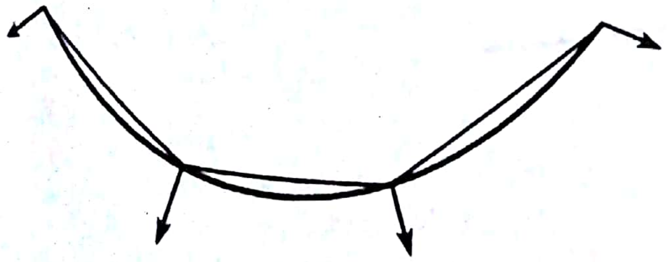

FIGURE 10-14
Use the direction of the true curve at each vertex of the approximation.

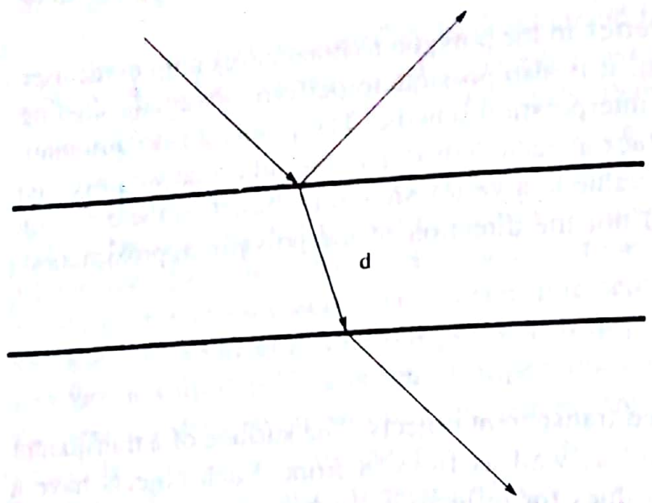

FIGURE 10-15
The absorption of light by an object depends on the length of the optical path within the object.

$$\frac{\sin \theta_1}{\sin \theta_2} = \frac{n_2}{n_1} \qquad (10.19)$$

The light ray is bent both when it enters a transparent object and when it leaves. To correctly show transparent objects, then, we have to retain back faces so that we can calculate the light path in and out of objects and also determine the length of the path within the object. This would be difficult to add to our system; however, there is a simple extension which can yield transparency effects.

If we assume that (1) for a given surface, the transparency coefficient for the object is a constant, (2) refraction effects can be ignored, and (3) no light source can be seen directly through the object, then the light coming through the object is just the transparency coefficient times the light coming from objects behind it. Our painter's algorithm overwrites objects in the back of the scene with the objects in the front. In effect, the light from the back object is replaced by that from the front object. But we can make the front object appear transparent if instead of blindly replacing old shading values, we multiply them by the transparency coefficient and add them to the values for the new surface. The shade actually displayed will correspond to light from the new object's surface plus the light behind, which is transmitted through the object.

$$v = v_r + t v_t \qquad (10.20)$$

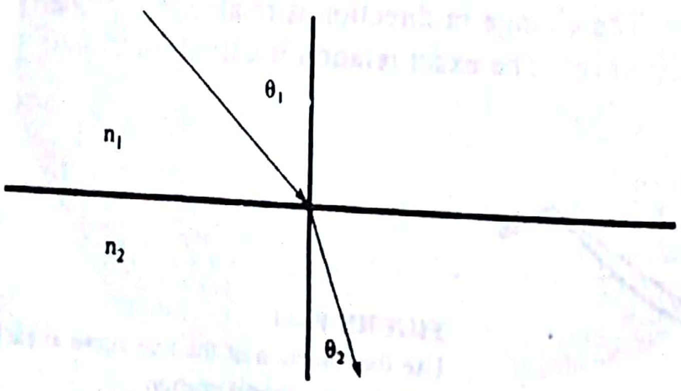

FIGURE 10-16
Refraction.

Here (v) is the total amount of light, (v_r) is the amount reflected from the surface as is given in Equation 10.16, (t) is the transparency coefficient, and (v_t) is the light coming from behind the object, which may be determined from the frame buffer value. We will require an inverse function for INTENSITY so that given the value in the frame buffer, we can determine the original SHADE-SETTING of the background object for use in the transparency calculation. Transparent objects often have a low reflectivity, and the reflected light (v_r) comes from specular reflection.

This simple approximation does not always yield realistic images, particularly where curved surfaces are being constructed. To improve it, we would like to include the angular behavior of the Fresnel reflection vs. transmission at the surface and also the attenuation due to thickness. Imagine light coming from some background object, passing through a transparent object, and reaching the eye. (See Figure 10-17.) If the transparent object has parallel sides (as do glasses, bottles, and windows), then the distance the light must travel through the object depends on the angle at which we view it. When viewed straight on, the distance the light travels within the object is less than when viewed at a glancing angle. Also, when the object's surface is perpendicular to the path of the light, the maximum amount is transmitted (instead of reflected). A simple approximation for this behavior is

$$t = (t_{max} - t_{min})\,(N \cdot E)^{\alpha} + t_{min} \tag{10.21}$$

This is the cosine of the angle between the eye and the surface normal raised to a power. The cosine is strongest when we view the surface straight on and drops off for glancing views. The power just enhances the effect. Values of α of 2 or 3 give reasonable effects.

REFLECTIONS

We have seen how to determine the effects of light coming directly from illumination sources, but in a realistic scene, every object can act as a light source. We should be able to see the specular reflections of objects, not just of lights. We look at a point p_1 on an object and ask what light is coming from that point and from where does it originate. We have seen how to find the light coming from diffuse illumination and directly from point sources. We can also follow a reflected ray back from our eye to the point and determine the incident ray. If we move back along the incident ray and reach a point p_2 on some other surface, then we know that the light (I) from this second point will be reflected at p_1. (See Figure 10-18.)

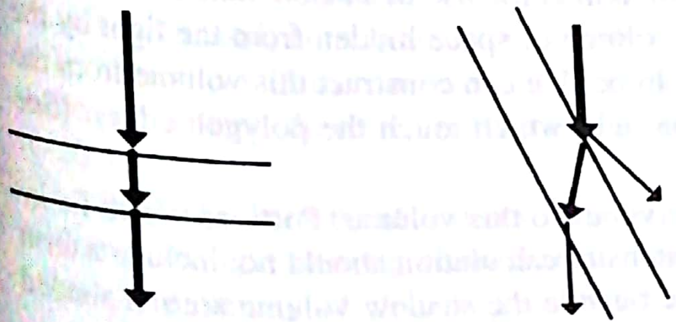

FIGURE 10-17
The reduction in light from reflection and absorption depends on the orientation.

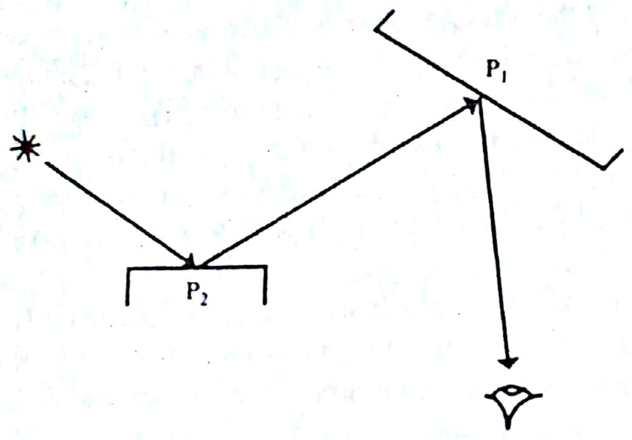

FIGURE 10-18
Light reflected from another object.

This will add a term to Equation 10.16 for the contribution of p_2 of the form

$$\frac{IF'/(E \cdot N)}{C + U} \qquad (10.22)$$

where F′ is calculated according to Equation 10.6 just as F, only c used in this calculation is given by

$$c = E \cdot N \qquad (10.23)$$

The light (I) for the point p_2 should be calculated as if the eye is located at p_1. For such calculations, all surfaces must be considered, even back faces (because a back face may be visible in a reflection). The method is discussed further in the section on ray tracing.

SHADOWS

The problem of shadows is very similar to the hidden-surface problem. A shadowed object is one which is hidden from the light source. Shadow and hidden-surface calculations are often performed together for efficient computation.

One approach to the shadow problem is to repeat the hidden-surface calculation using the light source as the viewpoint. This second hidden-surface calculation divides the polygons into shadowed and unshadowed groups. Surfaces which are visible and which are also visible from the light source are shown with both the background illumination and the light-source illumination. Surfaces which are visible but which are hidden from the light source are displayed with only the background illumination.

Another approach to the shadow problem is the use of *shadow volumes*. Consider a polygon and a light source. There is a volume of space hidden from the light by the polygon. This is the polygon's shadow volume. We can construct this volume from the polygon itself and from the rays from the light which touch the polygon edges. (See Figure 10-19.)

We can compare all other visible polygons to this volume. Portions which lie inside of the volume are shadowed. Their intensity calculation should not include a term from this light source. Polygons which lie outside the shadow volume are not shaded

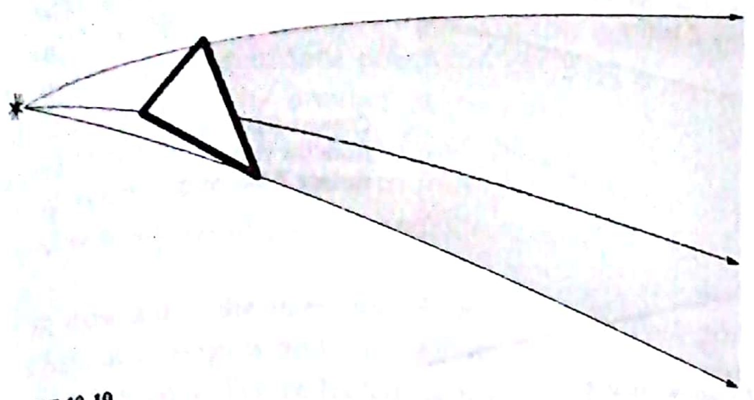

FIGURE 10-19
Shadow volume.

by this polygon, but might be shaded by some other polygon so they must still be checked against the other shadow volumes.

For simplicity, assume that all polygons are convex. (There are techniques for breaking up any polygon into convex components. In particular, polygons can be decomposed into triangles or trapezoids, which are always convex.) The shadow volume for a convex polygon is a convex object, which means that we can use the generalized clipping techniques to find shadowed areas. Points within the shadow volume are shadowed, and a point lies within the volume if it lies on the interior side of the polygon plane and all the shadow planes. So shadowing can be done by "clipping" all visible polygons to all possible shadow volumes.

The shadow volumes may be included as part of the hidden-surface processing. We include the surfaces of the shadow volumes along with the object surfaces. We include both front and back faces for shadow volumes, but tag them as such. We sort all polygons to determine their visibility. Shadow surfaces are invisible, and are not drawn but are counted. Where the number of front shadow faces in front of the visible polygon exceeds the number of back shadow faces, the polygon is in shadow.

We can use Z-buffer techniques to simplify the sorting. Suppose we have two Z buffers, one for determining the foremost visible surface and one for finding shadows. We begin with the Z-buffer hidden-surface removal algorithm described in Chapter 9. After this step, the visible surface Z buffer will contain the z coordinates of all visible points. The next step is to determine the shadow volumes for the scene. We can examine the surfaces in these shadow volumes to find which are the front and back faces. For each shadow volume, we can enter the z coordinate of the back-face pixels in the shadow Z buffer. We can also determine the z values for the pixels of the front faces on the shadow volume. Now if the z value of the front face of the shadow (which we just calculated) is in front of the visible surface (the z value in the visible surface Z buffer) and this in turn is in front of the back of the shadow volume (the z value in the shadow Z buffer), then the visible surface lies within the shadow volume. (See Figure 10-20.)

Shadows from point sources of light are sharp and harsh. In the real world we seldom find light coming from points. The sun and moon are disks; artificial light comes in a variety of shapes. Shadows from these finite shapes have softer edges.

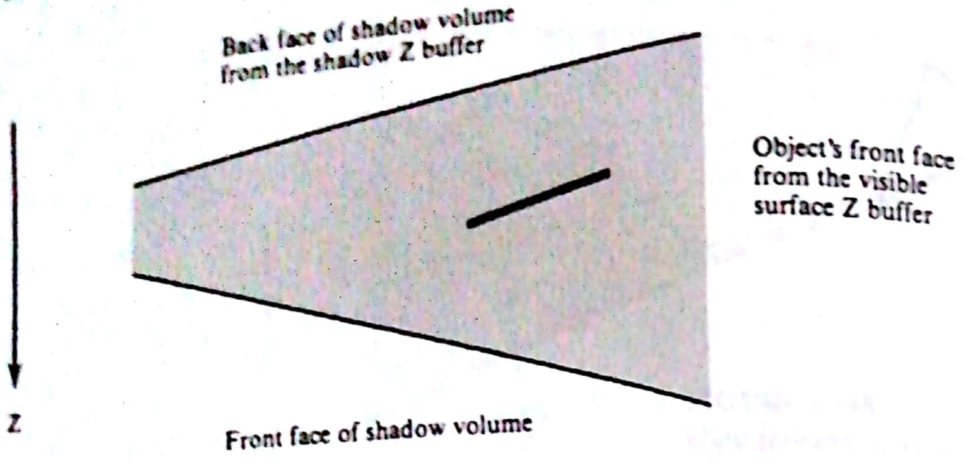
Back face of shadow volume from the shadow Z buffer
Object's front face from the visible surface Z buffer
Z
Front face of shadow volume

FIGURE 10-20
Using a shadow Z buffer to find shaded objects.

Much of the current work on shadows is in modeling these finite light sources to give more realistic results.

RAY TRACING

There is a simple brute-force technique which can yield startlingly realistic computer-generated images. (See Plate 15.) Its basic form is easy to implement and requires little of the machinery of the graphics system we have been constructing. It can handle curved surfaces as well as flat polygons. It includes perspective projection, hidden-surface removal, reflections, shadows, and transparency. The technique is called *ray tracing*, and the problem with it is that it is notoriously slow. The idea is to determine the intensity for each pixel one at a time by following a ray back from the viewpoint through the pixel and into the object space. We can compute the intersection of that ray with all the surfaces. If none is encountered, then the pixel has the background shade. If one or more surfaces are impacted by the ray, we find the one closest to the viewpoint. It is the one which will be seen (the rest are hidden by it). This, in effect, solves the hidden-surface problem for every pixel. (See Figure 10-21.)

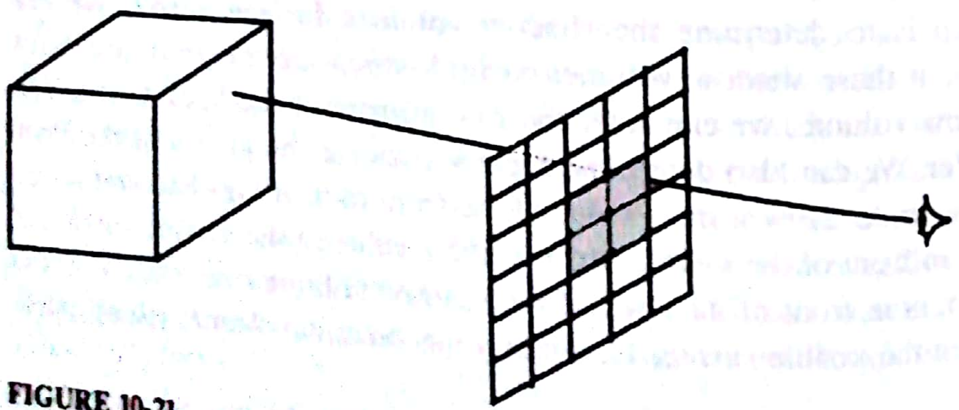

FIGURE 10-21
Ray tracing.

Next we can find the illumination of the surface point by calculations such as Equation 10.16. We may, however, as part of this calculation, look for the intersection of the rays between the surface point and the light sources with other objects in the scene. If a ray is cut by another surface, then the corresponding light source is shadowed for the pixel. This solves the shadow problem for every pixel. The same ray-following routines that were used to find the surface being drawn can be used to find shadows, only the starting point and direction of the ray are different. (See Figure 10-22.)

We must add to the intensity value for the surface point the amount of light reflected from other objects and, if the point belongs to a transparent object, the light transmitted through it. The reflected light is equal to the coefficient of specular reflection times the amount of light coming in along an incident ray. We find the light along this incident ray by recursively applying the method which we are describing. We imagine the surface point as the viewpoint and ask how much light comes to it along a given direction. We examine this new ray for intersections with objects and find the light from the closest. This may entail further recursive calls, until either the ray makes no intersections or the accumulated spectral reflection coefficients have reduced the light to where it is no longer significant. (See Figure 10-23.)

If objects are transparent, then we must also calculate and add in the transmitted light. As with the reflection case, we calculate the refracted ray which would give rise to the light along the ray we followed. We recursively apply our method to this ray to determine the light coming along it. We multiply the amount of light by the transparency and attenuation factors and add it to the total. (See Figure 10-24.)

Most of the time spent in ray tracing is in finding the intersection of a ray with the nearest surface. As the complexity of the image goes up, the number of surfaces which must be checked against the ray also increases. There is much interest and research into ray-tracing techniques, both in modeling the properties of objects and light

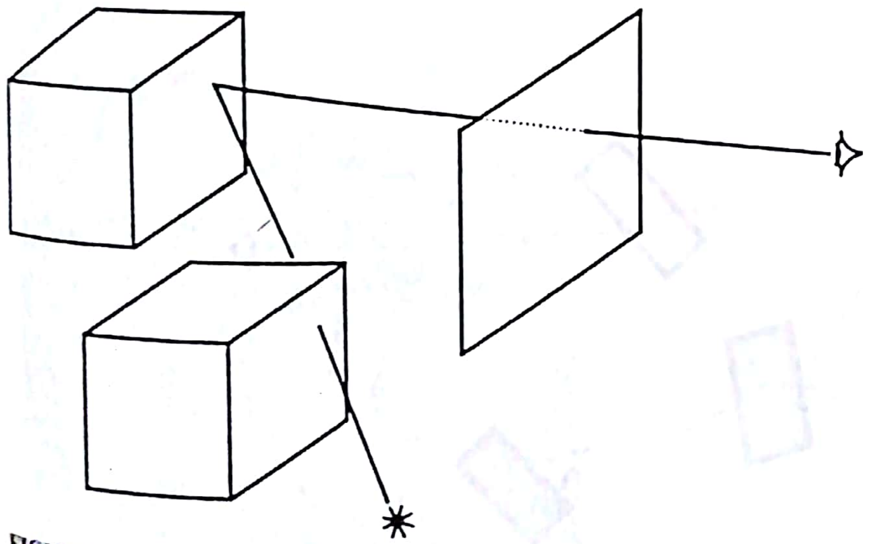

FIGURE 10-22
Ray tracing to find shadows.

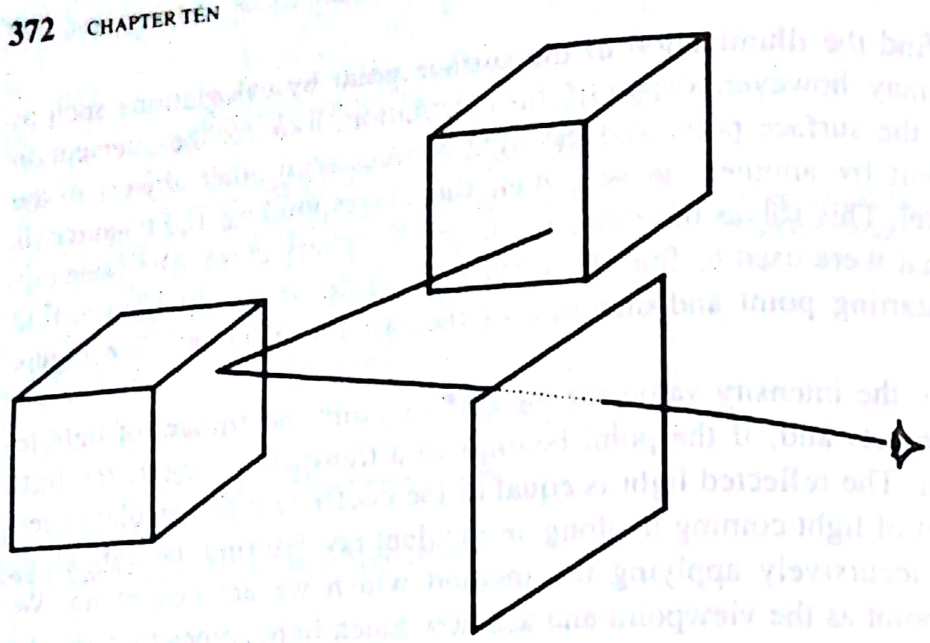

FIGURE 10-23
Ray tracing to find reflections.

sources to make them more realistic and in finding data structures and algorithms which make it more efficient.

HALFTONES

Many display devices allow only two imaging states per pixel (on or off, black or white). This is fine for line drawings, but when displaying surfaces, we would often like more variety than just black or white. There would be little point to the shading calculations we have discussed if there were no way to produce gray images. We discussed in Chapter 3 how our polygon-filling algorithm could be extended to display patterns. Using such patterns allows us to give different polygons different appear-

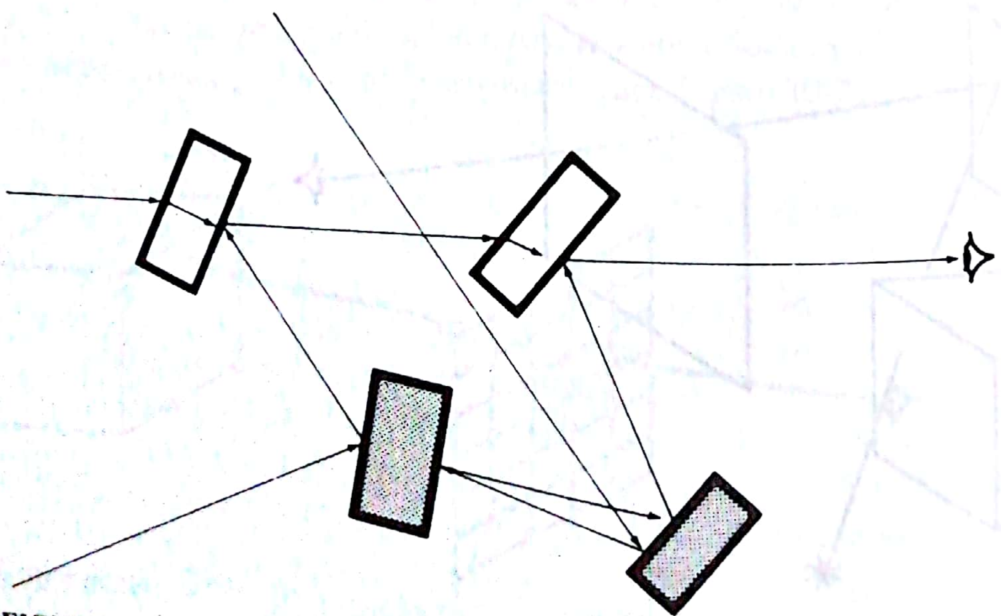

FIGURE 10-24
Repeated ray tracing.

aces, and with a judicious choice of patterns, this mechanism can also provide gray levels for use in shading. But for realistic images, as might be generated by ray tracing or might be input from a television camera or scanner, we must make further extensions so as to be able to control the apparent intensity in finer detail than we did for entire polygons. We have as a guide the work done by the printing industry. Photographs are routinely reproduced in books and newspapers using only two states (ink or no ink). The technique is called *halftoning*. A continuous tone image is shown through a screen. This produces a regularly spaced pattern of small dots. The dots vary in size according to the original image intensity; so where the image is light, the dots are small and the background color of the paper predominates. Where the image is dark, the dots are large and merge together to cover the paper with ink. (See Figure 10-25.)

How can we produce halftones? Our first try might be to divide our binary pixels into groups corresponding to the halftone cells, and call each group a single gray-level pixel. We could develop a set of dot patterns for the groups, making dot patterns for the group correspond to intensity values for the gray-level pixel. Every time we set a gray-level pixel to an intensity value, we would really be copying a binary dot pattern into the group of binary pixels which form it. For example, we could make each 2 × 2 pixel square our gray-level pixel. This would give us five possible intensity settings. We could design a pattern for each setting. (See Figure 10-26.)

Our effective resolution would be half of the binary resolution of the device, but we would have five intensity values instead of just two. (See Figure 10-27.)

The scheme we have outlined will work, but we can do better. We can produce images with a better effective resolution and still have as many gray levels. The reason this can be done is that there exists more than a single possible pattern for many of the intensity levels. Consider the intensity level of two active pixels for the 2 × 2 halftone cell. There are six possible patterns. (See Figure 10-28.)

The idea is to select patterns which closely match the structure of the image. For example, if we are looking at a polygon edge which crosses the halftone cell, as in Fig-

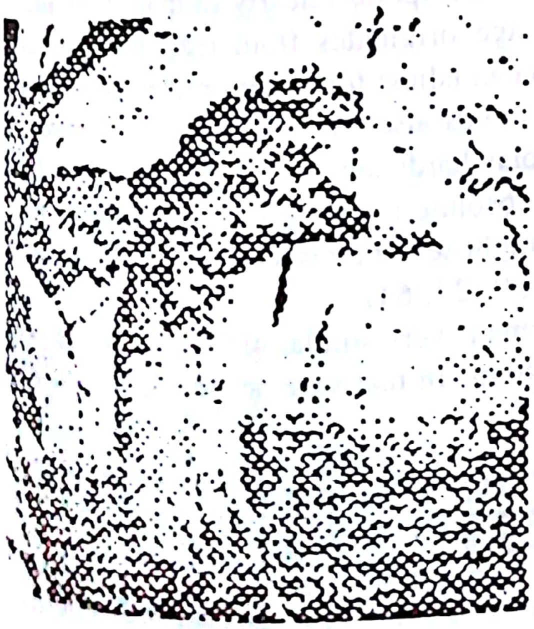

FIGURE 10-25
Enlarged halftone image of a bird in flight.

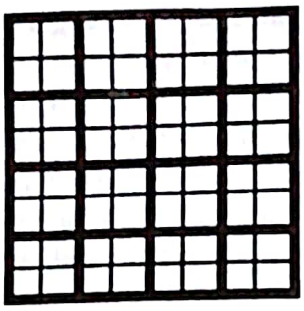

FIGURE 10-26
A grid of halftone cells.

ure 10-29, then for a dark polygon on a light background the pattern with the right two pixels shaded would be most appropriate. For a lighter polygon, perhaps only one of the two right pixels would be colored, but the left pixels should remain light. A simple method which comes close to implementing this kind of halftoning is an *ordered dither*. It is done by constructing a pattern of threshold values, called a *dither matrix*, which correspond to the individual pixels in the halftone cell. The pattern can be replicated to cover the screen so that every binary pixel has an associated threshold value. We then determine the intensity value for every pixel (full resolution, not just gray-level pixels). We compare the intensity of each pixel against its threshold value to determine whether the pixel is turned on or off.

This method also has the advantage of allowing intensity values to range over any scale of values desired. An image could be constructed in which each pixel has one of 256 possible intensity values. The image could be shown on a display using a halftone screen with 5 gray levels, or one with 50 gray levels, or one with 256 gray levels by just designing the corresponding cell of threshold values. For example, suppose we display such pixels using the five-level 2 × 2 cell. A possible dither matrix is shown in Figure 10-30. With this pattern, the lower right pixel would be turned on if it were set to an intensity greater than 50. The upper left pixel would be turned on only if it were set to an intensity greater than 200.

Notice that we can alter the appearance of the image by changing the threshold values. The intensity at the display need not correspond linearly to that of the source image. This is useful when the source image originates from real life, such as a scanned photograph. Altering the thresholds can adjust for under or over exposure and can be used to enhance the image. The thresholds also can be used to compensate for nonlinearities in either the scanning, the display hardware, or our eyes. Our eyes are in fact sensitive to intensity ratios rather than absolute intensities, so to achieve a set of gray levels which look equally spaced, we might select threshold values which change exponentially (1, 2, 4, 8, 16) or cubically (1, 9, 27, 64).

Implementation of this halftoning scheme is very similar to the method described in Chapter 3 for filling a polygon with a pattern. In that scheme we used the polygon

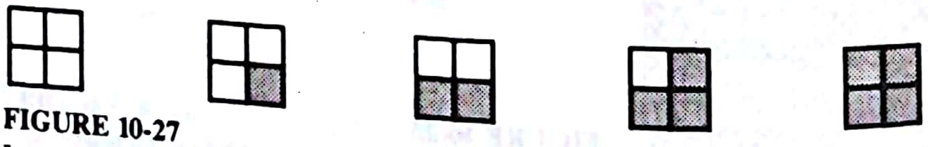

FIGURE 10-27
Intensity settings for the halftone cell.

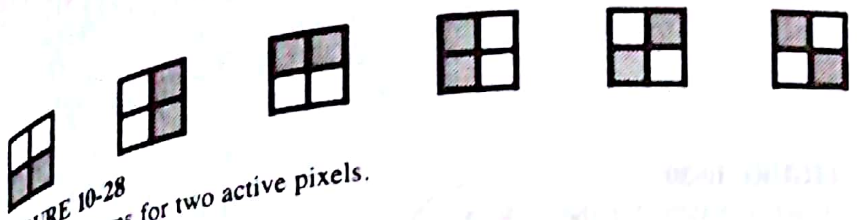

FIGURE 10-28
Possible patterns for two active pixels.

style to select a pattern and set the frame buffer pixel to the corresponding pixel element. For halftoning, when filling a polygon, we look up the threshold value for each pixel and compare it with the intensity value for the polygon. The results of the comparison are used to decide where to set the pixel. A replacement algorithm for FILLIN which does this halftoning is as follows:

10.12 Algorithm HALFTONE-FILLIN(X1, X2, Y) Fills in scan line Y from X1 to X2 using halftoning

Arguments	X1, X2 end positions of the scan line to be filled
	Y the scan line to be filled
Global	HALFTONE-CELL array of intensity thresholds
	FRAME the two-dimensional (binary) frame buffer array
	FILLCHR the intensity of the polygon
Local	X for stepping across the scan line
	HX, HY for accessing the halftone cell
Constants	HALF-X, HALF-Y the x and y dimensions of the halftone cell
	ON, OFF the possible pixel values

```
BEGIN
   IF X1 = X2 THEN RETURN;
   HX ← MOD(X1, HALF-X) + 1;
   HY ← MOD(Y, HALF-Y) + 1;
   FOR X = X1 TO X2 DO
      BEGIN
         IF FILLCHR > HALFTONE-CELL[HX, HY] THEN FRAME[X, Y] ← ON
         ELSE FRAME[X, Y] ← OFF;
         IF HX = HALF-X THEN HX ← 1
         ELSE HX ← HX + 1;
      END;
   RETURN;
END;
```

Note that the above halftoning scheme and the pattern filling of Chapter 3 can be combined to allow filling polygons with gray level and photographic style patterns (or

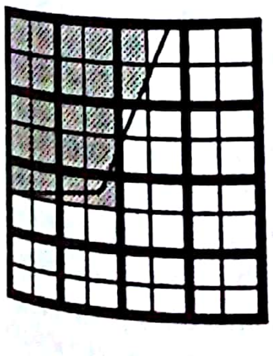

FIGURE 10-29
Select patterns to match the image shape.

200	150
100	50

FIGURE 10-30
A dither pattern for the 2 × 2 cell.

Gouraud shading) on a bilevel display. In the combined scheme we would look up both the threshold value from the halftone description and the intensity value from the pattern or polygon for the given pixel. These would then be compared to determine if the pixel should be set.

COLOR

Light is an electromagnetic wave and has wave properties. One of these properties is *frequency* (ν). The frequency describes how many cycles of the wave occur every second. A related property is the *wavelength* (λ), the distance in space between crests of the wave. They are related properties because light moves at a constant speed ($c = 3 \times 10^8$ meters/second in a vacuum). The speed must be the length of a cycle times how many cycles pass per second, so

$$c = \nu\lambda \tag{10.24}$$

Our eyes are able to detect differences in frequency (or wavelength) for a small range of the possible values. We can see light with wavelength between about 4×10^{-7} meters and 7×10^{-7} meters. Each wavelength appears to us as a color from violet at 400 nanometers(nm) through the rainbow to red at 700 nm. The reason we can see these colors is that our eyes have three different color sensors which are sensitive to different parts of the visible spectrum. They are called the blue, green, and red *photopigments*, and they give respective peak responses at blue, green, and yellow light, but also give lesser responses at other values so as to cover the entire visible range. (See Figure 10-31.)

We are not equally sensitive to all colors. Our eyes give the greatest response to green. We are less sensitive to red and blue light, and cannot see infrared or ultraviolet light at all. Because of this, the *luminosity* or perceived brightness of the light does not correspond directly to the energy of the light. It takes less energy to increase the brightness of green 1 *lumen* (a unit of brightness) than it takes to increase red 1 lumen, which is in turn less than the energy needed to increase the perceived intensity of blue light the same amount. (See Figure 10-32.)

Light with a single wavelength produces the sensation of a "pure" color. But note that there may be other ways to achieve the same response from the eye. For example, light at 490 nm excites all three photopigments, but the same effect could be achieved with two less intense lights, one at 450 nm to excite the blue and one at about 540 nm to excite the green and red photopigments. Thus blue-green can be seen either from a blue-green light or from a mixture of blue light and green light.

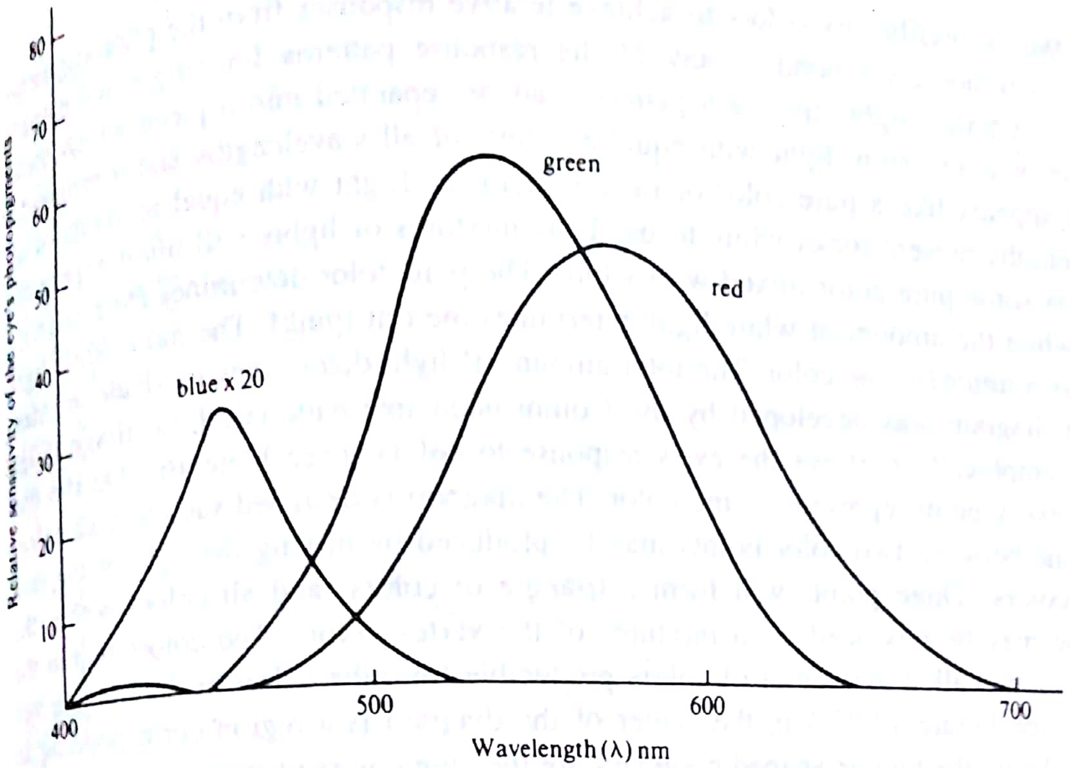

FIGURE 10-31
Relative response of the eye's photopigments.

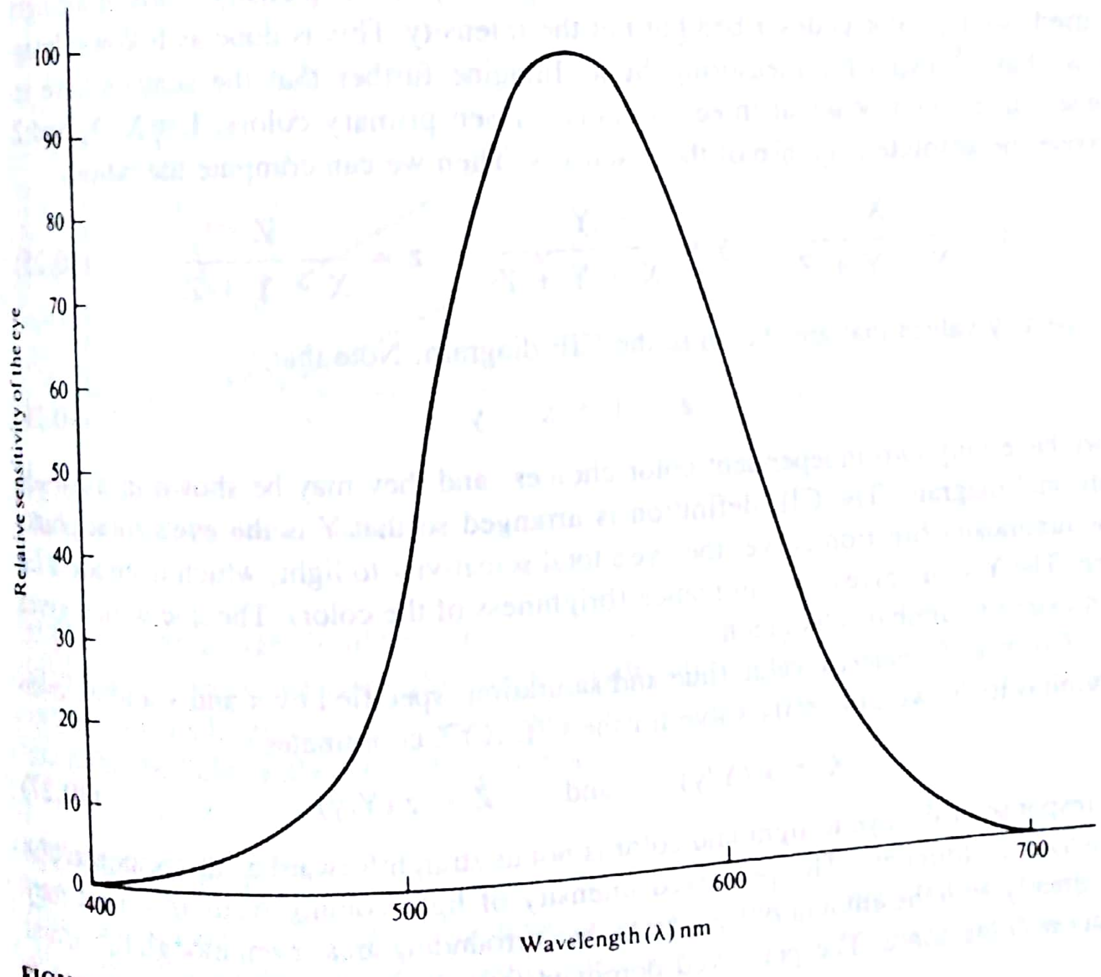

FIGURE 10-32
Total luminosity response of the eye.

We can easily mix colors to achieve relative responses from the photopigments which will not correspond to any of the response patterns for single wavelength (monochromatic) light. But such patterns can be separated into a piece which looks like the response from light with equal amounts of all wavelengths and a remainder which appears like a pure color or its complement. Light with equal amounts of all wavelengths present looks white to us. Thus mixtures of lights will often appear the same as some pure color mixed with white. The pure color determines the hue (e.g., red), while the amount of white light determines the tint (pink). The more white light the less *saturated* is the color. The total amount of light determines its shade or intensity. A diagram was developed by the Commission Internationale L'Eclairage (CIE) which graphically portrays the eye's response to colors. (See Plate 16.) On the diagram, every point represents some color. The diagram is designed such that all colors on a line between two color points may be produced by mixing the light of the end-point colors. Three points will form a triangle of colors, and all colors within the triangle may be produced from mixtures of the vertex colors. The colors within the triangles are called the gamut of colors producible from the colors at the triangle vertices. (See Figure 10-33.) In the center of the diagram is a region corresponding to white. Along the tongue-shaped boundary are the pure colors of monochromatic light. Across the base are the purples, which have no monochromatic representative.

The color of light is determined by the relative responses of the three photopigments in the eye. There is also the absolute response which tells us how much total light there is (how bright it is). In the CIE diagram, just the intensity ratios of the light are used, so the color is described but not the intensity. This is done as follows: Imagine we have sensors for measuring light. Imagine further that the sensors have response curves with peaks at three specially chosen primary colors. Let X, Y, and Z measure the absolute response of these sensors. Then we can compute the ratios

$$x = \frac{X}{X+Y+Z} \qquad y = \frac{Y}{X+Y+Z} \qquad z = \frac{Z}{X+Y+Z} \tag{10.25}$$

It is the x, y values that are shown in the CIE diagram. Note that

$$z = 1 - x - y \tag{10.26}$$

so we have only two independent color choices, and they may be shown in a two-dimensional diagram. The CIE definition is arranged so that Y is the eye's total photopic luminosity-function curve, the eye's total sensitivity to light, which is greatest for green. The Y value gives the luminance (brightness of the color). The hue which gives the maximum luminance is green.

Given a *chromaticity* value (hue and saturation) specified by x and y and an overall luminosity Y, we can easily solve for the CIE XYZ coordinates.

$$X = x\,(Y/y) \qquad \text{and} \qquad Z = z\,(Y/y) \tag{10.27}$$

The response of the eye to light and color is not as straightforward as the response of a physicist's instruments. The perceived intensity of light coming from an object may vary greatly with the amount of light from the surrounding area, even though the actual radiation is the same. The perceived dominant frequency or hue may appear to shift

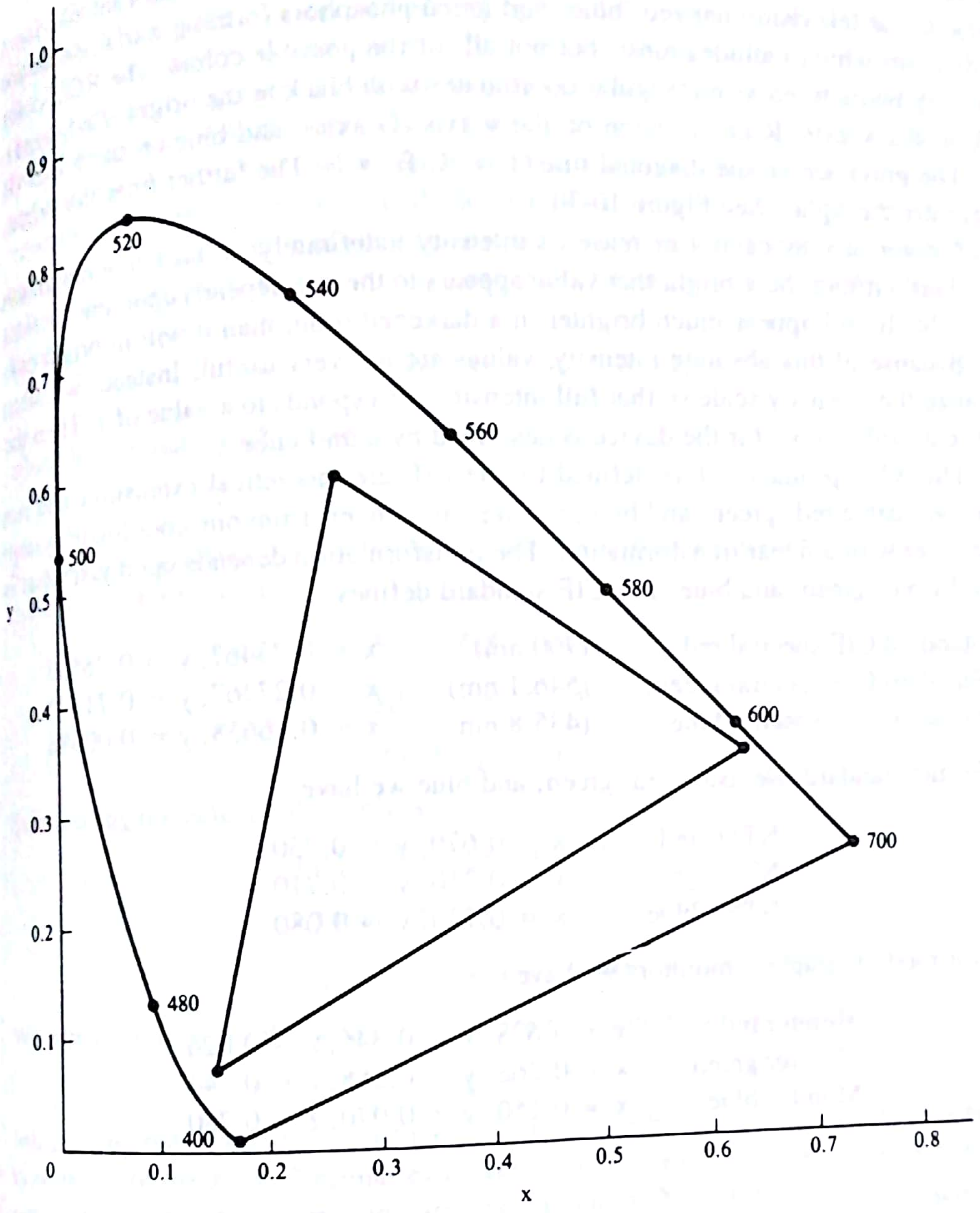

FIGURE 10-33
The color gamut for a color monitor.

slightly as white light is added. The perceived hue and saturation may also change with intensity, dropping out altogether for low radiation levels. In the remainder of the discussion we shall largely ignore these differences, giving rules for specifying the physical light and color rather than the psychophysical perception.

COLOR MODELS

There are several other ways to parameterize colored light. These are called color models. The *red-green-blue* (RGB) model is often encountered in computer graphics. It cor responds to the red, green, and blue intensity settings of a color monitor or television display. Color television has red, blue, and green phosphors forming a triangle on the CIE diagram which includes most, but not all, of the possible colors. The RGB color space may be pictured as rectangular coordinates with black at the origin. Red is measured on the x axis (R axis), green on the y axis (G axis), and blue on the z axis (B axis). The grays are on the diagonal line G = R, B = R. The farther from the origin, the brighter the light. (See Figure 10-34.)

A color display cannot increase its intensity indefinitely; it has some maximum value. Furthermore, how bright that value appears to the eye depends upon the ambient room light. It will appear much brighter in a darkened room than it will in bright sunlight. Because of this absolute intensity, values are not very useful. Instead, we often normalize the intensity scale so that full intensity corresponds to a value of 1. Then the set of realizable colors for the device is described by a unit cube.

The XYZ primary colors defined by the CIE are theoretical extensions and not really the visible red, green, and blue, but we can convert from one coordinate system to the other with a linear transformation. The transformation depends upon just what is meant by red, green, and blue. The CIE standard defines

Standard CIE spectral red	(700 nm)	x = 0.73467, y = 0.26533
Standard CIE spectral green	(546.1 nm)	x = 0.27367, y = 0.71741
Standard CIE spectral blue	(435.8 nm)	x = 0.16658, y = 0.00886

But for the standard television red, green, and blue we have

NTSC red	x = 0.670, y = 0.330
NTSC green	x = 0.210, y = 0.710
NTSC blue	x = 0.140, y = 0.080

And for modern graphics monitors we have

Monitor red	x = 0.628, y = 0.346, z = 0.026
Monitor green	x = 0.268, y = 0.588, z = 0.144
Monitor blue	x = 0.150, y = 0.070, z = 0.780

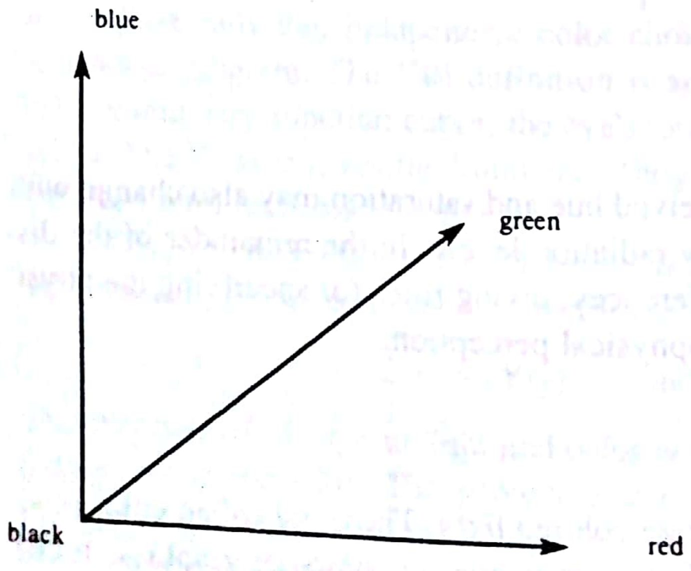

FIGURE 10-34
The RGB color space.

Usually only x and y values are listed because we can find the z values using Equation 10.26. These values are not enough to tell us how to convert between RGB and XYZ. We know the colors but not their relative strengths. If we just pick three colors and add them together, we are unlikely to get white; but we want to weight or scale the primaries we have chosen so that if we do select equal amounts of the red, green, and blue, we do come out with white. The amount to scale depends on just what we mean by white, and there are several choices (the color of an electric light, the color of sunlight at noon, the color of the overcast sky, and others). Monitors are often factory realigned to the color of a black body heated to 6500 degrees Kelvin (K). This has CIE values

$$x = 0.313, \qquad y = 0.329 \tag{10.28}$$

which for luminancy of 1 give

$$X_w = 0.950, \qquad Y_w = 1.000, \qquad Z_w = 1.086 \tag{10.29}$$

So if we were considering the monitor primaries, we would have to supply the scaling factors r, b, g such that

$$\begin{aligned} x_r r + x_g g + x_b b &= X_w \\ y_r r + y_g g + y_b b &= Y_w \\ z_r r + z_g g + z_b b &= Z_w \end{aligned} \tag{10.30}$$

Substituting the actual values gives

$$\begin{aligned} 0.628r + 0.268g + 0.150b &= 0.950 \\ 0.346r + 0.588g + 0.070b &= 1.000 \\ 0.026r + 0.144g + 0.780b &= 1.089 \end{aligned} \tag{10.31}$$

We can solve this set of linear equations for r, g, and b. Doing so gives

$$r = 0.758, \qquad g = 1.116, \qquad b = 1.165 \tag{10.32}$$

Multiplying the chromaticities of the primary colors by these weighting factors tells us how much of the X, Y, Z primaries is contributed by each of the R, G, B primary colors. For example, for the red primary, the amount of X contributed is

$$X_r = (0.628)(0.758)R \tag{10.33}$$

The total X amount is the sum of the contributions from the R, G, and B components of a color.

$$\begin{aligned} X &= x_r rR + x_g gG + x_b bB \\ &= (0.628)\,(0.758)R + (0.268)\,(1.116)G + (0.150)\,(1.165)B \\ &= 0.476R + 0.299G + 0.175B \end{aligned} \tag{10.34}$$

Similar statements may be made about Y and Z. We can perform the multiplications and combine the three equations into a matrix expression. The result is a transformation for converting from the RGB coordinates of the monitor to XYZ coordinates.

$$[X\ Y\ Z] = [R\ G\ B]\begin{vmatrix} 0.476 & 0.262 & 0.020 \\ 0.299 & 0.656 & 0.161 \\ 0.175 & 0.082 & 0.909 \end{vmatrix} \tag{10.35}$$

Finding the inverse matrix, we can write

$$[R\ G\ B] = [X\ Y\ Z]\begin{vmatrix} 2.750 & -1.118 & 0.138 \\ -1.149 & 2.026 & -0.333 \\ -0.426 & 0.033 & 1.104 \end{vmatrix} \tag{10.36}$$

Hard-copy devices may use cyan, magenta, and yellow as the primary colors rather than red, green, and blue. These are the colors of the inks which are placed on the white paper. They begin with white and use the inks to absorb and remove some colors. Cyan removes red, magenta absorbs green, and yellow removes the blue. This complementary relationship between red, green, blue, and cyan, magenta, yellow makes it easy to convert from one color model to the other. Assuming units where 1 means full color and 0 means no color, and assuming that the cyan, magenta, and yellow colors are exact complements of red, green, and blue, then

$$C = 1 - R, \qquad M = 1 - G, \qquad \text{and} \qquad Y = 1 - B \tag{10.37}$$

or

$$[C\ \ M\ \ Y] = [1\ \ 1\ \ 1] - [R\ \ G\ \ B]$$

We can also picture this relation. Start with the unit cube of all colors, with black at a corner which serves as the origin of the RGB coordinate system. The R, G, and B axes along the edges of the cube meet at the black corner. The opposite corner on the cube will correspond to white, and will serve as the origin for the CMY coordinates. The C, M, and Y axes on the edges of the cube meet at the white corner. Each of the other corners of the cube can be labeled with one of the primary colors. (See Figure 10-35.)

Printers often use the four colors cyan, magenta, yellow, and black. In theory, only the three inks cyan, magenta, and yellow (CMY) should be needed. Mixing the three should produce an ink which absorbs all the light, yielding black; but in practice, the inks may not absorb completely or mix well, so a fourth black ink is used to set the shade.

Color television specifies colors by the YIQ color model. The Y is the same as the Y in the CIE primaries and corresponds to the luminosity or brightness. It is the Y component which is displayed on a black-and-white television. The I and Q coordinates determine the hue and saturation. The transformation from RGB coordinates is as follows:

$$[I\ Q\ Y] = [R\ G\ B]\begin{vmatrix} 0.60 & 0.21 & 0.30 \\ -0.28 & -0.52 & 0.59 \\ -0.30 & 0.32 & 0.11 \end{vmatrix} \tag{10.38}$$

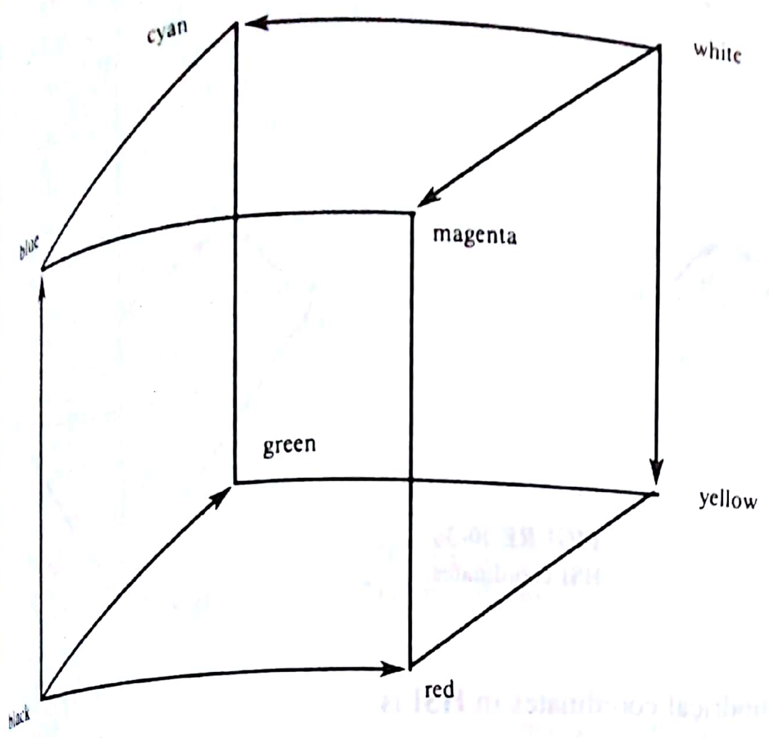

FIGURE 10-35
The RGB-CMY color cube.

Human vision is much more sensitive to intensity than it is to color. For this reason, we do not need to provide as much color information as intensity information. We can quantize the I and Q coordinates in bigger steps (fewer bits) and give fine detail (more bits) to the Y coordinate. This allows an overall reduction in the amount of information which must be sent to achieve a high-quality image. The YIQ model allows a more efficient representation of a color image intended for a human observer.

Another color model is the hue, saturation (or chroma), and intensity(brightness) model (*HSI*). The *hue* or *tone* is the pure color of the light. It is the dominant color we see. *Saturation* indicates how much white light is mixed with the color (the tint). Saturation of 0 means no color; we see white or gray. The intensity tells how much total light there is (the shade). Hue, saturation, and intensity are often plotted in cylindrical coordinates with hue the angle, saturation the radius, and intensity the axis. (See Figure 10-36.)

The conversion between HSI and RGB models can be done in two steps. To convert from RGB to HSI we first rotate the coordinates so that the third axis is along G = R, B = R line. This gives a rectangular coordinate version of the HSI space. The second step is a change to cylindrical coordinates and proper scaling of the intensity. The coordinates of the rectangular HSI space will be labeled M_1, M_2, I_1 where I_1 is the intensity and M_1, M_2 describe the hue-saturation plane. (See Figure 10-37.)

The transformation is

$$[M_1\ M_2\ I_1] = [R\ G\ B] \begin{vmatrix} 2/\sqrt{6} & 0 & 1/\sqrt{3} \\ -1/\sqrt{6} & 1/\sqrt{2} & 1/\sqrt{3} \\ -1/\sqrt{6} & -1/\sqrt{2} & 1/\sqrt{3} \end{vmatrix} \tag{10.39}$$

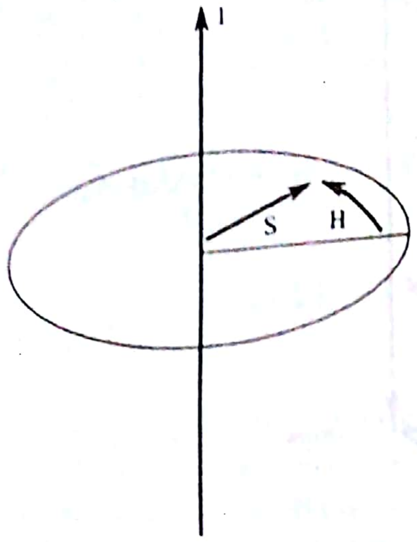

FIGURE 10-36
HSI coordinates.

The conversion to cylindrical coordinates in HSI is

$$\begin{aligned} H &= \arctan(M_1 / M_2) \\ S &= (M_1^2 + M_2^2)^{1/2} \\ I &= I_1 \sqrt{3} \end{aligned} \tag{10.40}$$

Conversion from HSI to RGB takes the inverse steps

$$\begin{aligned} M_1 &= S \sin H \\ M_2 &= S \cos H \\ I_1 &= I/\sqrt{3} \end{aligned} \tag{10.41}$$

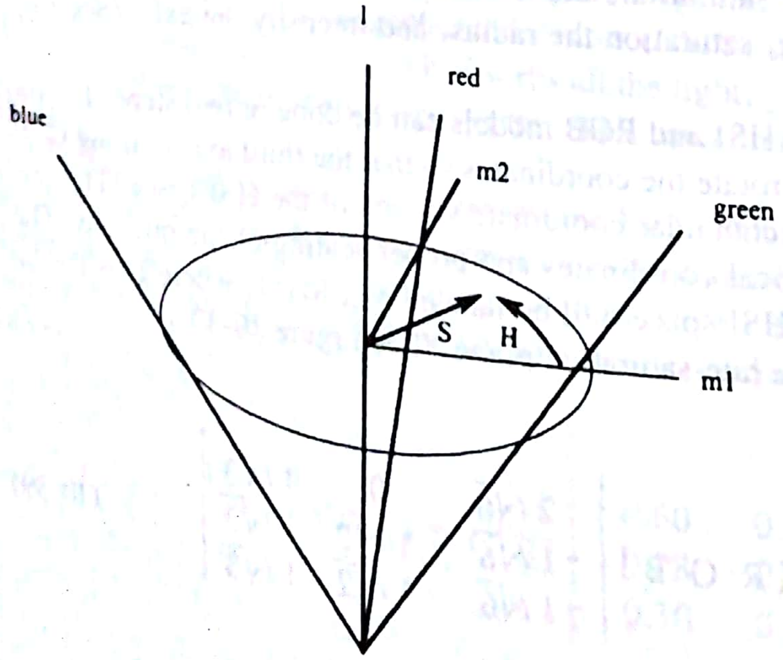

FIGURE 10-37
Correspondence between RGB and HSI coordinates.

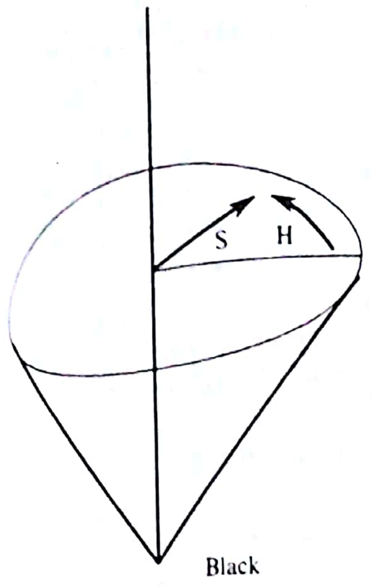

FIGURE 10-38
The cone of possible colors.

$$[\,R\ G\ B\,] = [\,M_1\ M_2\ I_1\,]\begin{vmatrix} 2/\sqrt{6} & -1/\sqrt{6} & -1/\sqrt{6} \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{vmatrix} \qquad (10.42)$$

When dealing with inks on paper, we sometimes use a color space with cylindrical coordinates where the angle is hue, brightness is along the axis, and the radius is how much of the colored ink is on the paper. This gives a cone of colors. At the point is black, along the axis is saturation 0 (grays), and along the surface of the cone is the most saturated color. As you move up or down, the amount of black ink changes. Moving out or in adds or removes color from the portion of the paper not covered by black. The radial saturation measure is usually normalized so that it is 1 on the cone's surface. A value of 1 then specifies the maximum saturation achievable by the inks for the given brightness. (See Figure 10-38.)

Sometimes hue, saturation, and value (HSV) are used to specify colors. Value is the intensity of the maximum of the red, blue, and green components of the color. Value has the same "feel" as brightness (as value decreases, the shade looks darker) but is easier to calculate. Also to make the calculation easier, the hue can be approximated by the distance along the edge of a hexagon. The relation between HSV and RGB is given by the next two algorithms:

10.13 Algorithm RGB-TO-HSV(R, G, B, H, S, V) Converts from red, green, blue values in the range 0 to 1 to hue, saturation, and value
Hue is expressed in degrees

```
Arguments  R, G, B the red, green, blue coordinates
           H, S, V for return of the hue, saturation, and value
Local      R1, G1, B1, how relatively close the color is to red, green, and blue
           X intensity of the minimum primary color
BEGIN
   H ← 0;
```

```
    find dominant primary color
    V ← MAX(R, MAX(G, B));
    find the amount of white
    X ← MIN(R, MIN(G, B));
    determine the normalized saturation
    S ← (V − X) / V;
    IF S = 0 THEN RETURN;
    R1 := (V − R) / (V − X);
    G1 := (V − G) / (V − X);
    B1 := (V − B) / (V − X);
    in which section of the hexagon does the color lie?
    IF R = V THEN
       IF G = X THEN H ← 5 + B1
       ELSE H ← 1 − G1
    ELSE IF G = V THEN
       IF B = X THEN H ← R1 + 1
       ELSE H ← 3 − B1
    ELSE
       IF R = X THEN H ← 3 + G1
       ELSE H ← 5 − R1;
    convert to degrees
    H ← H * 60
    RETURN;
END;
```

The algorithm first finds the value parameter, which is simply the intensity of the dominant primary color. Next, it finds the minimum primary color. This can be mixed with equal amounts of the other two colors to form white and is therefore a measure of how much white is in the color. The expression V − X is the amount of the dominant color left after part is used to make the white component of the color. The relative strength of this net color becomes our normalized saturation measure. If the saturation is 0 then there is no hue (we have a gray), so the algorithm returns with a default hue of 0. The R1, G1, and B1 variables are used to tell how much of the second most important color there is relative to the dominant color. The IF statements then find the dominant color and move the hue away from it by this amount.

10.14 Algorithm HSV-TO-RGB(H, S, V, R, G, B) Converts hue, saturation, value coordinates to red, green, blue

```
Arguments  H, S, V the hue, saturation, and value
           R, G, B for return of the red, green, blue coordinates
Local      H1 the hue in units of 1 to 6
           I integer part of the H1 hue, indicates the dominant color
           F fractional part of the H1 hue, used to determine second color
           A an array that holds the three color values while deciding which is which
BEGIN
    convert from degrees to hexagon section
    H1 ← H / 60;
    find the dominant color
    I ← INT[H1];
    F ← H1 − I;
    A[1] ← V;
    A[2] ← V;
    A[3] ← V * (1 − (S * F));
    A[4] ← V * (1 − S);
    A[5] ← A[4];
    A[6] ← V * (1 − (S * (1 − F)));
    map strengths to rgb
    IF I > 4 THEN I ← I − 4 ELSE I ← I + 2;
    R ← A[I];
    IF I > 4 THEN I ← I − 4 ELSE I ← I + 2;
    B ← A[I];
    IF I > 4 THEN I ← I − 4 ELSE I ← I + 2;
    G ← A[I];
    RETURN;
END;
```

In this algorithm the integer part of the hue determines the dominant color. Its intensity V is saved in the first and second array cells. The fractional part describes the second most important color. Its value is in either the third or sixth array cell, depending on which side of the dominant color it lies. The weakest color intensity is saved in the fourth and fifth cells. Then the values are copied from the array to the R, G, B parameters according to which color was dominant (the original I).

Because of the simple way that HSV calculates hue, the colors are mapped onto a hexagonal pyramid or hexcone instead of a circular cone. (See Figure 10-39.) A variation on the HSV model called the hue, lightness, and saturation model (HLS) is used by Tektronix. It deforms the hexcode by stretching the white point along the value axis. This forms two hexcones placed base to base, with black at one apex and white at

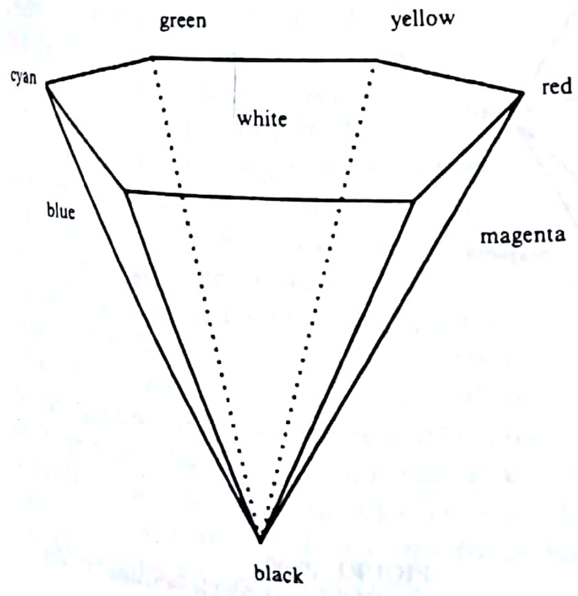

FIGURE 10-39
The HSV model hexcone of color.

the other. In this model, black has a lightness of 0 and white a lightness of 1, but fully saturated colors have a lightness value of only 0.5. The lightness is defined as the average of the maximum and minimum RGB values, which is (V + X)/2 in the notation of algorithm 10.13. The saturation is (V − X)/(V + X) if the lightness is less than 0.5, and (V − X)/(2 − V − X) for lightness greater than 0.5. (See Figure 10-40.) The HSI, HSV, and HLS color models are useful in implementing programs where the user must select colors. It is not always easy to tell which RGB values are needed to get a desired color. The HSV model corresponds to how an artist mixes colors: selecting a hue, adding white to get the proper tint, and adding black to get the desired shade. It is more intuitive. We might allow the user to specify a color using the HSV model and then convert it to RGB for display. Or the user might select a color from a displayed palette which would be converted to HSV units for later modification.

GAMMA CORRECTION

The above transformations are based on the fact that light behaves linearly, that the effect of light from two sources is simply the sum of the individual sources. While this is true for light, it probably won't be true for the voltages applied to a monitor to produce

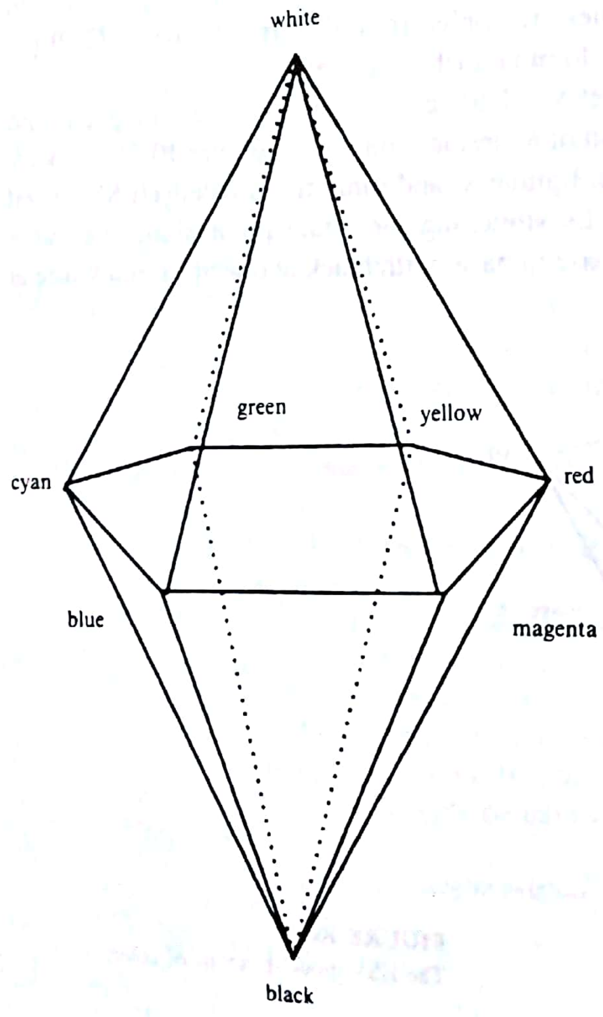

FIGURE 10-40
The HLS model double hexcone of color.

the light. Doubling the voltage at the monitor will probably not yield twice the light. Instead, there is typically a power relation.

$$I = k V^{\gamma} \tag{10.43}$$

The exponent γ is usually around 2.

Suppose we determine the value (I) for an intensity we would like to display. We still must determine the proper voltage setting for the display hardware to give this intensity. To do this, we will have to solve Equation 10.43 for (V).

$$V = \left(\frac{I}{k}\right)^{1/\gamma} \tag{10.44}$$

This is called *gamma correction*. One fast way of performing gamma correction is to construct a table of the supported intensity values and corresponding display voltages. Gamma correction then becomes a simple table lookup. For a color display, gamma correction must be applied to each of the three primary colors. Some monitors provide internal circuitry to perform the gamma correction and thus appear to be linear devices.

COLOR TABLES

Some display devices are capable of generating a large variety of colors. For example, if a device has 8 bits of control over each of the red, green, and blue channels, then each channel can have 256 intensity levels, and together 16,777,216 different colors may be produced. To fully specify all these colors we would need 24 bits per pixel in the frame buffer memory. Since we can usually get by with far fewer colors, this is not very cost-effective. An alternative is a color table. A *color table* allows us to map between a color index in the frame buffer and a color specification. Suppose our frame buffer had 8 bits per pixel. This would allow us only 256 colors; but what if we use the frame buffer entry as an index to access a color table. Suppose the color table has 256 entries, each entry containing a 24-bit color selection. Then we can show any 256 of the 16,777,216 possible colors. And by changing the values in the color table, we can change the available color selection to contain the 256 colors we most desire for the current scene. (See Figure 10-41.)

Note that changing a value in the color table can alter the appearance of large portions of the display (any part of the display which referenced that entry). Because it is the hardware which makes these changes, they usually occur very fast. Some clever techniques have been devised which take advantage of this hardware feature. For example, we might give the screen a background color from color-table entry 1 and draw a complex object using color-table entry 2. Now if table entry 2 has the same value as table entry 1, then the object will be invisible; it will be indistinguishable from the background. But if we change table entry 2 to some other color, the object will suddenly appear. Changing its color can make it visible or invisible, and this change is often far faster than we can draw or erase the object. Drawing or erasing requires changing a lot of pixel values in the frame buffer, while changing the color only requires changing a single color entry.

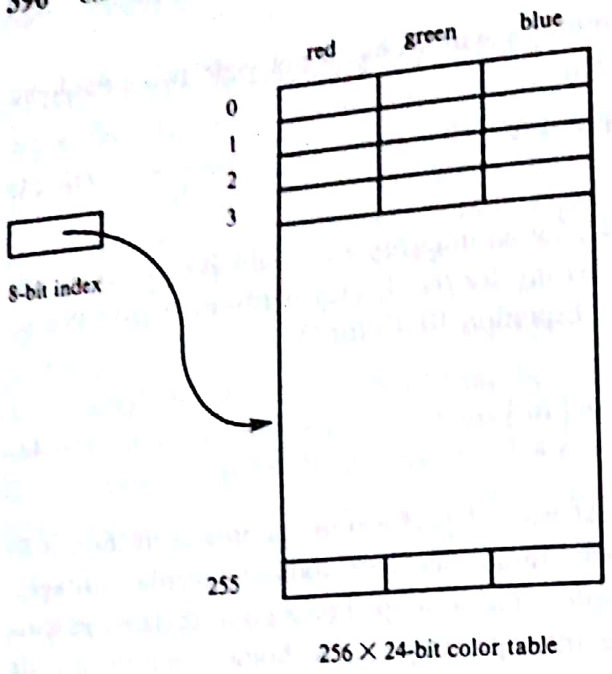

FIGURE 10-41
A color table.

We can have several invisible objects (as many as there are color-table entries). By sequencing through visibility changes, animation effects can be achieved.

Color tables have been used to give false color or pseudocolor to gray-scale images. They have also been used for gamma correction, lighting and shading, and color model transformations.

EXTENDING THE SHADING MODEL TO COLOR

As we mentioned at the start of the chapter, objects have a color-dependent index of reflection which can be represented by three reflectivity values: one for red, one for green, and a third for blue. Our simple model for specular reflection is not color-dependent, but the light sources can be colored. Background light and point sources must be given three components. As a result, our shading expression of Equation 10.16 splits into three equations, one for each color component.

$$v_r = B_r R_r + \sum_j \frac{P_{rj}[R_r(L \cdot N) + D_j G_j F/(E \cdot N)]}{C + U_j}$$

$$v_g = B_g R_g + \sum_j \frac{P_{gj}[R_g(L \cdot N) + D_j G_j F/(E \cdot N)]}{C + U_j} \tag{10.45}$$

$$v_b = B_b R_b + \sum_j \frac{P_{bj}[R_b(L \cdot N) + D_j G_j F/(E \cdot N)]}{C + U_j}$$

FURTHER READING

An introduction to the problems of realistic image generation is given in [NEW77]. First attempts at shading just used the distance from the viewpoint to determine brightness [WYL67]. An early model for shading is described in [PHO75], and an alternative computation method for it was given in [DUF79]. This model was improved in [BLI77]; it is the one used in this text. The model was further extended to include colored specular reflections in [COO82]. Extensions for light sources with intensity distributions and color are given in [WAR83]. Hardware extensions to the frame buffer have been devised to allow fast scan conversion, hidden-surface removal, clipping, smooth shading, shadows, and transparency. The method, which is based on all pixels being able to evaluate linear expression in parallel, is described in [FUC85]. The idea of imaging transparent objects by modifying the painter's algorithm was presented in [NEW72]. Improved transparency approximations are discussed in [KAY79]. Gouraud presented smooth shading by interpolation in [GOU71]. A comparison of shadow algorithms is given in [CRO77]. Shadows are also discussed in [NIS85*b*]. Hidden-surface techniques are used in [ATH78] to find polygons that are not in shadow. In [WIL78] shadows are determined by first computing a Z buffer using the light source as the viewpoint. Then as each point is calculated for display, it is compared against that buffer to find out if anything lies in front relative to the light source to cause a shadow. This allows curved shadows on curved objects. An extension of scan-line algorithms to include shadows is given in [BOU70]. The method of generating shadows by an extended Z buffer is described in [BRO84]. Ray tracing was first suggested in [APP68]. Modeling of light and ray tracing are discussed in [WHI80]. An introduction to ray tracing and its use with constructive solid geometry are given in [ROT82]. A system for modeling, ray tracing, and display of three-dimensional objects is described in [HAL83]; the system uses an improved illumination model including transparency. Ray tracing for shadows and reflection of diffuse radiation are described in [NIS85*a*]. Ray-tracing research includes methods for antialiasing and also for the elimination of sharp edges for shadows, motion blur, and translucency. See [AMA84], [COO84], [HEC84], and [LEE85]. There is also research into ray tracing of objects other than polygons. See [BLI82], [BOU85], [KAJ83*a*], [KAJ83*b*], [TOT85], and [VAN84]. And there is interest in improving the speed of ray tracing. See [AMA84], [GLA84], [HEC84], [PLU85], and [WEG84]. Ray tracing can reproduce the specular reflection of objects by objects, but does not model the interaction of diffuse radiation between objects. This is considered in [GOR84] and [COH85]. Our system allows us to shade surfaces composed of polygons of uniform color, but real objects can have textures and variations of surface color. One approach to this problem is to map texture patterns onto the surface. See [BLI76] and [CAT80]. Another is to define a three-dimensional texture pattern which can be accessed for surface points [PEA85]. Texture patterns can be used to provide perturbations to the surface normal as well as the reflectivity. This can give surfaces a rough or wrinkled appearance. See [BLI78] and [HAR84]. Reviews of techniques to obtain continuous tone images from bilevel devices are given in [ALG83] and [KNO72]. Halftoning with an ordered dither is described in [JAR76], [JUD74], [KLE70], and [LIP71]. Sometimes the halftone pattern is oriented at an