Lighting Models in Computer Graphics

In this short document, we take a brief look at some of the initial ideas basic to the problem of shading elements of a scene to realistically simulate the appearance of illuminated colored objects. This is an enormous and sophisticated topic, ranging from optical principles of basic physics to the human psychology of visual perception. Sophisticated approaches must take into consideration not just physical principles of light absorption and reflection from surfaces and materials of various sorts, but the the way in which the human mind perceives relationships between intensity of illumination and various properties of color denoted by terms such as hue, saturation, tint, shade, tone, and so on. All of these issues have been studied in considerable detail, and most standard computer graphics references devote considerable space to the subject.

The limited time available in this course restricts our discussion here to the most basic aspects of illumination – analysis of the amount of light reflected from a surface element to the eye of the observer. We will give only passing consideration to issues of color, essentially by simulating brighter colors using higher intensities of pure RGB components.

Figure 1 to the right illustrates the physical elements of illumination of which we will have to take account. The goal will be to model the "appearance" at the observer's position of the surface element shown. We will assume we are working with a planar surface element. To get the basic formulas, we will also initially assume that there is at most a single point-size source of identifiable illumination, and that the observer is also representable as a point. As mentioned earlier, we will be looking only at intensity of illumination here, without taking any account of color.

Figure 1

A number of directions or vectors are defined in Figure 1:

- $\mathbf{u}_n$ is a unit vector in the direction perpendicular to the surface, the so-called **unit normal vector**
- $\mathbf{u}_s$ is a unit vector in the direction from the point **P** towards a point light source illuminating **P**
- $\mathbf{u}_o$ is a unit vector in the direction from point **P** to the location of the observer
- $\mathbf{u}_r$ is a unit vector in the direction of an outgoing reflected ray of light which originated at the point source.

We will construct these vectors when the need for each arises.

The simplest of lighting models take account of three illumination effects: **ambient light**, **diffuse reflection**, and **specular reflection**. What the observer "sees" is considered to be the sum of these three effects.

1. Ambient Light

Ambient light is the general illumination in the scene, not associated or attributable to any particular source of light. It is the physical result of light that has reflected from a succession of many objects in the room and so has lost any directionality (the jargon is that it is *isotropic* – the same, *iso*, for every direction, *tropic*). This is the light in a room that allows you to see the wad of used chewing gum on the underside of your table-chair top. In the model we adopt, the intensity of light reaching the observer due to ambient light reflection from the surface is represented as

$$I_{ambient} = I_a k_a \qquad \text{(LIGHT-1)}$$

where I_a is a constant giving the intensity of the ambient illumination in the scene (call it the *ambient light intensity* if you like), and k_a is a constant (with value between 0 and 1) indicating the fraction of ambient illumination of the surface which is reflected by the surface (you could call this the *ambient light reflection coefficient*). For very reflective surfaces, k_a will be near 1, whereas for very unreflective surfaces, k_a will be near zero. This factoring into an illumination intensity and a reflectance takes account of the fact that even in a fairly bright scene, nonreflective surfaces will appear dimmer than highly reflective surfaces. From a computer graphics point of view, the constant k_a should be viewed as a sort of adjustable "fudge" factor. Although the principle behind it sounds simple, there is no simple physical or material property of a real object that would allow us to calculate in advance what value to give to k_a. As with the other "reflection coefficients" introduced below, the value of k_a is selected quite arbitrarily to achieve a desired visual effect.

2. Diffuse Reflection

The diffuse reflection term in the model attempts to account for the physical mechanism whereby the surface absorbs light from an identifiable (point) source, and re-emits only certain wavelengths of that light uniformly in all directions.

It is this emission of only certain components of the light that gives the appearance of the surface being colored. Thus, a surface appears to be blue because all other color components of the illuminating light have been absorbed by the surface, and so the observer sees only "blue wavelengths". That's why when a blue object is illuminated with light having wavelengths only in some other color range (say red), the blue object appears black – there is no blue light in the illumination, hence no blue light can be re-emitted. If no light is emitted, the surface appears black.

Since the emitted light has the same intensity in all directions (because of the absorption/re-emission mechanism being modelled), the intensity of the emitted light depends only on the intensity of illumination of the surface. But now, since the illumination is identified with a specific light source, it is no longer the same for all surface orientations.

Figure 2 to the right illustrates what happens. The upper part of the figure represents a beam of light coming in from the left, and striking a surface at an angle. Defining the so-called angle of incidence, θ, as the angle between the incoming light direction and the direction perpendicular (or **normal**) to the surface, we see that the beam must actually illuminate an extent of the surface measuring not d units wide, but d/cos θ units wide. The lower part of the figure illustrates the trigonometry involved.

Figure 2

Thus, if the illumination provided by the point source is I_p, then the illumination per unit area of the surface must be $I_p \cos\theta$. Allowing for the fact that not all of the light striking the surface may be reflected by introducing a diffuse reflection coefficient, we adopt the following formula for modelling diffuse reflection:

$$I_{diffuse} = I_p k_d \cos\theta \qquad \text{(LIGHT-2)}$$

You might think at first that this same sort reduction in perceived intensity by a cosine factor must occur at the observer's position, since the observer may be seeing the reflected light at an angle as well. However, in viewing the reflecting surface at an angle, the observer will also be viewing a larger

element of surface per unit of viewing angle, and the two effects cancel out. (That is, the observer still sees the entire reflecting surface, and so sees all of the light reflecting off of it in his direction.)

Note that the greatest diffuse illumination occurs when $\theta = 0^0$ or $\cos\theta = 1$ is at its maximum value – that is, when the light source is immediately behind the observer. When $\theta = 90^0$, the light source is edge on to the surface and (since $\cos 90^0 = 0$), we get zero illumination. Formula (LIGHT-2) is restricted to values of θ between 0^0 and 90^0. Larger angles imply the light is behind the surface, in which case no illumination should occur.

We can do slightly better than (LIGHT-2) here by taking into account the actual locations of the point light source and the point **P**. Let (x_s, y_s, z_s) be the coordinates of the light source, and let (x, y, z) stand for the coordinates of point **P**. Then the direction from **P** to the light source is given by the vector

$$\mathbf{s} = (x_s - x, y_s - y, z_s - z) \qquad \text{(LIGHT-3a)}$$

This vector has length[1]

$$d_s = \sqrt{(x_s - x)^2 + (y_s - y)^2 + (z_s - z)^2} \qquad \text{(LIGHT-3b)}$$

and so the unit vector, $\mathbf{u}_s$, in the direction from the point **P** to the light source is given simply by

$$\mathbf{u}_s = \left(\frac{x_s - x}{d_s}, \quad \frac{y_s - y}{d_s}, \quad \frac{z_s - z}{d_s} \right) \qquad \text{(LIGHT-3c)}$$

These formulas may not look all that pretty, but the various quantities in them can be calculated from available information very straightforwardly.

We also need to be able to compute the components of the unit normal vector, $\mathbf{u}_n$. This vector is determined by the orientation of the surface in space, and so will have to come from some mathematical specification of that surface. The document in this series that deals with viewing transformations gives a method for calculating unit normal vectors for polygonal surfaces for which we know coordinates of the vertices. Another approach will be illustrated by an example involving spheres later in this document, in which the unit normal vector can be obtained from the known mathematical properties of the surface. For the moment, we'll assume that $\mathbf{u}_n$ is known.

Now, given the way $\mathbf{u}_s$ and $\mathbf{u}_n$ are defined in Figure 1, the angle θ is just the angle between these two unit vectors. Furthermore, we know from the basic properties of vectors that the angle between two vectors, **a** and **b** is given by the formula

$$\cos\theta = \frac{\mathbf{a} \bullet \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}$$

where $\mathbf{a}\bullet\mathbf{b} = a_x b_x + a_y b_y + a_z b_z$ is the dot product of the two vectors, and $|\mathbf{a}|$ and $|\mathbf{b}|$ are the lengths, respectively, of the two vectors. Usually, one would use this formula to compute $\cos\theta$ and then use the inverse cosine function to obtain the value of θ. However, in this case, we actually need the value of $\cos\theta$ itself. Furthermore, since our two vectors have unit length, the two factors in the denominator will just be 1's, and so can be dropped. Thus, we end up with

$$\cos\theta = \mathbf{u}_n \bullet \mathbf{u}_s$$

and so, (LIGHT-2) becomes

[1] We will be making fairly extensive use of very basic properties of vectors here. If this material seems to be moving a bit fast, it might be a good idea to do a quick review of the course document on vectors.

$$I_{diffuse} = I_p\, k_d\, \mathbf{u}_n \bullet \mathbf{u}_s \qquad \text{(LIGHT-4)}$$

as the effective working formula for the diffuse reflection contribution to our lighting model.

3. Specular Reflection

The specular reflection gives rise to the bright shiny highlight seen when a smooth or glossy curved surface is illuminated by a light source. This reflection occurs from the thin transparent layer giving the object its glossy coating, and so has the same color (or lack of color) as the light source. Because it really amounts to a mirror-like reflection of the light (rather than an absorption and re-emission in all directions), the greatest intensity of the specular reflection will be along the direction, $\mathbf{u}_r$, of the reflected ray (see Figure 1).

Figure 3 to the right shows the the four relevant directions schematically. The direction of $\mathbf{u}_r$ is determined by the well known optical principle that the angle of reflection (the angle between $\mathbf{u}_r$ and $\mathbf{u}_n$) must equal the angle of incidence (the angle between $\mathbf{u}_s$ and $\mathbf{u}_n$), both denoted by θ in the figure. The symbol α denotes the angle between $\mathbf{u}_r$ (the direction along which the specular reflection should have maximum intensity) and the actual direction of the observer, $\mathbf{u}_o$.

Figure 3

A simple but effective way to model specular reflection was proposed by Phong Bui-Tong in 1975. He proposed to model the intensity of the specular reflection using a term of the form

$$I_{specular} = I_p\, k_s\, (\cos\alpha)^n \qquad \text{(LIGHT-5)}$$

Since we're talking about the same light source that results in the diffuse reflection term, the illumination intensity, I_p, is the same as in (LIGHT-2) above. k_s is, of course, the specular reflection coefficient, indicating the fraction of incident light that is reflected specularly, and is often given a value closer to 1 than to zero. It is the final factor, $(\cos\alpha)^n$ which gives the required rapid fall-off of intensity of reflection as the observer's direction differs more and more from the direction of the reflected ray. The exponent n typically is given a value anywhere from 5-10 for very broad fuzzy specular highlights, up to 100 or 200 for very sharp narrow highlights.

When $\alpha = 0$, $\cos\alpha = 1$, and so $(\cos\alpha)^n = 1$. For any value of $\alpha > 0$, we have $\cos\alpha < 1$, and so $(\cos\alpha)^n$ will be even more less than 1. The higher the value of the exponent n, the faster will be the fall-off in value of $(\cos\alpha)^n$, as illustrated by Figure 4. In the figure, the horizontal axis corresponds to α ranging from 0^0 to 90^0, and the successive curves, starting from the right, are for n = 1, 2, 5, 10, 50, and 100, respectively.

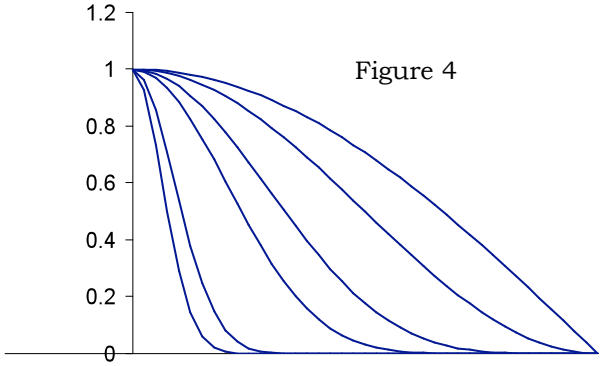

Figure 4

In order to turn (LIGHT-5) into a convenient working formula, we need to come up with a way to calculate α or at least $\cos\alpha$. Following the discussion leading to (LIGHT-4), we realize that

$$\cos\alpha = \mathbf{u}_r \bullet \mathbf{u}_o \qquad \text{(LIGHT-6)}$$

Indicating the coordinates of the observer by (x_o, y_o, z_o), and using the generic symbols (x, y, z) for the coordinates of the point **P**, we have that

$$\mathbf{u}_o = \left(\frac{x_o - x}{d_o}, \frac{y_o - y}{d_o}, \frac{z_o - z}{d_o} \right) \qquad \text{(LIGHT-7a)}$$

where

$$d_o = \sqrt{(x_o - x)^2 + (y_o - y)^2 + (z_o - z)^2} \qquad \text{(LIGHT-7b)}$$

The calculation of $\mathbf{u}_r$ is a bit more involved, but not too difficult to obtain using basic properties of vectors. Figure 5 to the right illustrates the situation. Here the heavy dot is the point **P**, and $\mathbf{u}_s$, $\mathbf{u}_n$, and $\mathbf{u}_r$ have the usual meanings. The physical principle that the angle of incidence must equal the angle of reflection of the light ray in question is satisfied by the equality of the two angles labelled θ. But if those two angles are equal, then so are the two angles labelled φ in the diagram, since $\varphi + \theta = 90^0$ in both cases. But, since the two arrows, labelled $\mathbf{u}_s$ and $\mathbf{u}_r$ are both of length 1, the two right triangles of which these arrows form the hypotenuses must be congruent (the same size and shape). Thus, the sides labelled **m** are of equal length, and the sides labelled respectively **a** and **–a** are of equal length, as the symbols in each case are indicating. The dotted arrows indicate the decomposition of $\mathbf{u}_s$ and $\mathbf{u}_r$ into horizontal and vertical components.

Figure 5

Now, in vector notation,

$$\mathbf{u}_r = \mathbf{m} + (\mathbf{-a}) = \mathbf{m} - \mathbf{a} \qquad \text{(LIGHT-8a)}$$

and

$$\mathbf{u}_s = \mathbf{m} + \mathbf{a} \quad \text{or} \quad \mathbf{a} = \mathbf{u}_s - \mathbf{m} \qquad \text{(LIGHT-8b)}$$

Thus,

$$\mathbf{u}_r = \mathbf{m} - (\mathbf{u}_s - \mathbf{m}) = 2\mathbf{m} - \mathbf{u}_s \qquad \text{(LIGHT-8c)}$$

But, as well, since the **u**'s are all unit length vectors, we can write that

$$\mathbf{m} = (\mathbf{u}_s \bullet \mathbf{u}_n)\, \mathbf{u}_n. \qquad \text{(LIGHT-9)}$$

That is, **m** is just the projection of $\mathbf{u}_s$ along the direction $\mathbf{u}_n$. Substituting (LIGHT-9) into (LIGHT-8c) then gives

$$\mathbf{u}_r = 2(\mathbf{u}_s \bullet \mathbf{u}_n)\mathbf{u}_n - \mathbf{u}_s \qquad \text{(LIGHT-10)}$$

expressing $\mathbf{u}_r$ completely in terms of $\mathbf{u}_s$ and $\mathbf{u}_n$. We use this formula to compute $\mathbf{u}_r$ required in (LIGHT-6), to get the value of $\cos \alpha$ required in formula (LIGHT-5). Note that the signs of terms in the formulas (LIGHT-8) – (LIGHT-10) depend on the orientation in which $\mathbf{u}_s$ and $\mathbf{u}_r$ are defined, and so may be different in other references that defined either of these vectors to be pointed in the opposite directions.

As an aside, note that formula (LIGHT-10) can be demonstrated to obey the requirement that the angle of incidence equal the angle of reflection. Since the **u** vectors are all unit length by construction, we know that the cosine of the angle between $\mathbf{u}_s$ and $\mathbf{u}_n$ is just $\mathbf{u}_s \bullet \mathbf{u}_n$ and the cosine of the angle between $\mathbf{u}_r$ and $\mathbf{u}_n$ is just $\mathbf{u}_r \bullet \mathbf{u}_n$. But

$$\mathbf{u}_r \bullet \mathbf{u}_n = [\, 2(\mathbf{u}_s \bullet \mathbf{u}_n)\mathbf{u}_n - \mathbf{u}_s \,] \bullet \mathbf{u}_n = 2(\mathbf{u}_s \bullet \mathbf{u}_n)\mathbf{u}_n \bullet \mathbf{u}_n - \mathbf{u}_s \bullet \mathbf{u}_n = 2(\mathbf{u}_s \bullet \mathbf{u}_n) - \mathbf{u}_s \bullet \mathbf{u}_n = \mathbf{u}_s \bullet \mathbf{u}_n$$

where we've made use of (LIGHT-10) to substitute for $\mathbf{u}_r$ in the first step, and later, the fact that $\mathbf{u}_n \bullet \mathbf{u}_n = 1$, since $\mathbf{u}_n$ is a unit vector. Thus $\mathbf{u}_r \bullet \mathbf{u}_n = \mathbf{u}_s \bullet \mathbf{u}_n$, and so cosines of the two angles in question are identical, and as long as $\mathbf{u}_s$ is not perpendicular to $\mathbf{u}_n$, this implies that the two angles are equal.

Example

Before putting all of the above results together into one master formula, we give a short example involving actual numbers. Figure 6 shows a sphere of radius 21 units, centered on the origin of the coordinate system. We will suppose the following physical coordinates of point **P**, the light source, and the observer:

P:	(16, 8, 11)
source:	(5, 20, 30)
observer:	(40, -10, 25)

Figure 6

Thus, the point **P** is near the middle of the right, front, upper octant of the sphere. The source is somewhat back, somewhat right, and well above the point **P**. The observer is quite far in front of, and a bit to the left and above the point P. Since the source is right, up, and back of **P**, we expect that the reflected ray will be left and forward of **P**, and probably downward.

Now from (LIGHT-3a), the vector **s** is

$$\mathbf{s} = (5 - 16, 20 - 8, 30 - 11) = (-11, 12, 19) \qquad \text{(LIGHT-11a)}$$

The length or magnitude of this vector is

$$d_s = \sqrt{\left((-11)^2 + (12)^2 + (19)^2\right)} = \sqrt{626} \cong 25.02$$

Thus, by (LIGHT-3c), we get here that

$$\mathbf{u}_s = \left(\frac{-11}{\sqrt{626}},\ \frac{12}{\sqrt{626}},\ \frac{19}{\sqrt{626}}\right) \cong \left(-0.4396,\ \ 0.4796,\ \ 0.7594\right) \qquad \text{(LIGHT-11b)}$$

rounded to four decimal places. Note that this is a vector pointing backwards, rightwards and upwards, as expected from the description of the relative positions of **P** and S above.

A sphere has the nice property that a line through its center always intersects its surface at right angles. Thus, the vector

$$\mathbf{v} = (16 - 0, 8 - 0, 11 - 0) = (16, 8, 11) \qquad \text{(LIGHT-11c)}$$

giving the displacement between **P** and the origin of the coordinate system at (0, 0, 0) will be perpendicular to the surface of the sphere at point **P**. To get $\mathbf{u}_n$, we just have to normalize this vector. First, its length, d_v, is

$$d_v = \sqrt{(16)^2 + (8)^2 + (11)^2} = \sqrt{441} = 21$$

(because, of course, being a point on the surface of a sphere of radius 21, **P** must be 21 units of length away from the center of the sphere). Thus,

$$\mathbf{u}_n = \left(\frac{16}{21},\ \frac{8}{21},\ \frac{11}{21}\right) \cong (0.7619,\ \ 0.3810,\ \ 0.5238) \qquad \text{(LIGHT-11d)}$$

again rounded to four decimal places. This is a vector pointing forwards, rightwards, and upwards, which is what we'd expect of a vector perpendicular to the right, front, upper octant of a sphere.

Now that we have $\mathbf{u}_s$ and $\mathbf{u}_n$, we can calculate the value of $\cos\theta = \mathbf{u}_s \bullet \mathbf{u}_n$ required for the diffuse reflection term, (LIGHT-4):

$$\cos\theta = \mathbf{u}_s \bullet \mathbf{u}_n = (\mathbf{u}_s)_x\,(\mathbf{u}_n)_x + (\mathbf{u}_s)_y\,(\mathbf{u}_n)_y + (\mathbf{u}_s)_z\,(\mathbf{u}_n)_z$$

$$= (-0.4396)(0.7619) + (0.4796)(0.3810) + (0.7594)(0.5238)$$

$$\cong 0.2455 \qquad \text{(LIGHT-11e)}$$

Finally, for the specular reflection term, we first need a vector in the direction from **P** to the observer:

$$\mathbf{v}_o = (40 - 16,\ -10 - 8,\ 25 - 11) = (24,\ -18,\ 14) \qquad \text{(LIGHT-11f)}$$

The magnitude of this vector is

$$d_o = \sqrt{(24)^2 + (-18)^2 + (14)^2} = \sqrt{1096} \cong 33.11$$

Thus,

$$\mathbf{u}_o = \left(\frac{24}{\sqrt{1096}},\ \frac{-18}{\sqrt{1096}},\ \frac{14}{\sqrt{1096}}\right) \cong (0.7249,\ \ -0.5437,\ \ 0.4229) \qquad \text{(LIGHT-11g)}$$

We also need $\mathbf{u}_r$, which is given by formula (LIGHT-10). We already know $\mathbf{u}_s \bullet \mathbf{u}_n \cong 0.2455$ from (LIGHT-11e), and so, substituting into (LIGHT-10) component by component, we get:

$$(\mathbf{u}_r)_x = 2(\mathbf{u}_s \bullet \mathbf{u}_n)\,(\mathbf{u}_n)_x - (\mathbf{u}_s)_x \cong 2(0.2455)(0.7619) - (-0.4396) \cong 0.8138$$
$$(\mathbf{u}_r)_y = 2(\mathbf{u}_s \bullet \mathbf{u}_n)\,(\mathbf{u}_n)_y - (\mathbf{u}_s)_y \cong 2(0.2455)(0.3810) - (0.4796) \cong -0.2926$$
$$(\mathbf{u}_r)_z = 2(\mathbf{u}_s \bullet \mathbf{u}_n)\,(\mathbf{u}_n)_z - (\mathbf{u}_s)_z \cong 2(0.2455)(0.5238) - (0.7594) \cong -0.5022$$

or

$$\mathbf{u}_r \cong (0.8138,\ -0.2926,\ -0.5022) \qquad \text{(LIGHT-11h)}$$

Thus, formula (LIGHT-6) gives us for this point,

$$\cos\alpha = \mathbf{u}_r \bullet \mathbf{u}_o$$
$$\cong (0.8138)(0.7249) + (-0.2926)(-0.5437) + (-0.5022)(0.4229)$$
$$\cong 0.5366$$

(LIGHT-11i)

(Notice that this means that $\mathbf{u}_o$ makes an angle of about 57.5^0 with $\mathbf{u}_r$, so the observer in this example is not really very close to the path of the reflected ray. This would mean that point **P** will not appear very shiny even if the sphere has a glossy surface.).

This completes the calculation of all the major pieces of the lighting model discussed earlier for this particular point and situation.

♦♦♦

Distance Attenuation

So far we haven't taken any account of the distance between the light source and the illuminated surface element. We would normally expect that objects located farther from a light will be illuminated less intensely than objects nearer the light source, and so assuming a constant intensity, I_p, for all surface elements will not be realistic.

The simplest physical model of how intensity varies with distance from the light source is to think of the source being at the center of a spherical shell and illuminating its entire inner surface. Since the area being illuminated increases in proportion to the square of the radius, but the total amount of light generated by the source stays the same, we expect the actual intensity of illumination to decrease in inverse proportion to the distance:

$$I_p \sim 1/d^2$$

This turns out not to work too well in computer graphics work, probably for a combination of several reasons, both physical and physiological. Instead, a formula has been devised which compromises between the unrealistic constant intensity and unrealistic inverse square intensity, usually written as

$$f_{att} = \min\left(\frac{1}{c_0 + c_1 d + c_2 d^2}, 1\right) \qquad \text{(LIGHT-12)}$$

Here, c_0, c_1, and c_2 are user specified constants for a particular light source and d is the distance between the light source and the illuminated surface element. When d is very small, the intensity of the light is $1/c_0$, approximately constant. When d is very large, the d^2 term in the denominator dominates, so the intensity is approximately $1/c_2d^2$, an approximately inverse square model. The min(_, 1) is just to make sure that due to a poor choice of these user specified constants, we never end up with f_{att} being bigger than 1 (which would mean more light is falling on the surface than leaves the light source!).

To take account of distance attenuation of illumination, we would then replace the factors I_p in formulas (LIGHT-2) and (LIGHT-5) with the product $I_p f_{att}$.

Putting It All Together (Including Multiple Light Sources)

We assume that the total illumination of a surface element is just the sum of the illuminations due to ambient lighting, diffuse reflection, and specular reflection, and thus get the overall formula

$$I = I_a k_a + I_p f_{att} [\, k_d (\mathbf{u}_s \bullet \mathbf{u}_n) + k_s (\mathbf{u}_r \bullet \mathbf{u}_o)^n \,] \qquad \text{(LIGHT-13)}$$

If there are two or more light sources, the illumination I would just be the sum of the right-hand sides of (LIGHT-13) for each source.

In these formulas, we've said nothing about color (which corresponds to the wavelength of the light from the source). In fact, the constants I_p, k_d, k_s, and perhaps the others in (LIGHT-13) as well, will have different values for different wavelengths (or colors) of light, and so this formula really only applies to a single color at a time. For highly sophisticated work, you would need to calculate the illuminations (LIGHT-13) for every relevant light wavelength involved, and then use that whole set of illuminations to determine the color with which the surface element is rendered. In the examples discussed below, we will assume light sources are a fixed pure color (red, green, blue, or white – though white as such is not a color), and so avoid this issue entirely. Discussions of ways of characterizing and working with color are beyond the scope of this course and so for further details you'll have to consult some of the standard references listed in the course outline.

A More Involved Example: Lighting a Sphere

A somewhat more involved example of the above formulas is provided by the problem of illuminating a sphere in three dimensions. The sphere is quite a simple shape to handle because its outline shape is just a circle, and its geometric properties are quite simple. To make things even simpler, we'll assume here that the drawing region of the screen is the z = 0 plane, and that world or object coordinates for the points on the sphere are the same as pixel coordinates on the screen (as far as x and y are concerned).

Figure 7

Figure 7 shows the coordinate system we'll use, and also the outline or silhouette of the sphere to be shaded. We consider it to be centered at the point $(x_c, y_c, 0)$ and of radius r. Further, we assume that the single light source is at location (x_s, y_s, z_s), and the observer is at (x_0, y_0, z_0), with reference to the same coordinate system as the sphere. For this example, we will use a simple orthographic projection, since it gives a realistic image of an actual sphere in three-dimensions.

The easiest way to organize the computations here is to scan the image, pixel-by-pixel, using a nested for-type loop:

```
for (x = xc – r; x <= xc + r; x++) {
        for (y = yc – r; y <= yc + r; y++)
        {
        …
        …
        }
}
```

in a sort of c-like code. This code looks as if x and y are integer quantities, because we're assuming they represent pixel coordinates on the screen. In actual implementations of this example, you will have to pay a bit more attention to the specific data types used for the various quantities.

The equation of this sphere is

$$(x - x_c)^2 + (y - y_c)^2 + z^2 = r^2 \qquad \text{(LIGHT-14)}$$

Thus, each of the $(2r)^2$ pixels forming the dotted square in Figure 7 can be classified in three ways:

if $(x - x_c)^2 + (y - y_c)^2 > r^2$, then the pixel is not part of the image of the sphere (since in this case, z^2 would have to be negative, which is impossible). These are the pixels inside the square, but outside the image of the sphere and so can be ignored.

if $(x - x_c)^2 + (y - y_c)^2 = r^2$, then we have a point which is just on the outline of the sphere, corresponding to z = 0.

if $(x - x_c)^2 + (y - y_c)^2 < r^2$, then we must have that z^2 is positive – in fact

$$z^2 = r^2 - [(x - x_c)^2 + (y - y_c)^2] \Rightarrow z = \sqrt{r^2 - (x - x_c)^2 - (y - y_c)^2} \qquad \text{(LIGHT-15)}$$

giving a point on the sphere in front of the xy-plane. (We can ignore the negative square root above, since those would be points on the back surface of the sphere, which aren't visible to us.)

With these three rules, we can easily associate with each pixel on the screen that is part of the image of the sphere a specific set of (x, y, z) coordinates on the surface of the actual sphere.

Now, a sphere has the well-known property that any line through the center is perpendicular to the surface of the sphere. Thus, at the point (x, y, z) on the surface of a sphere centered at (x_c, y_c, z_c), the vector ($x - x_c$, $y - y_c$, $z - z_c$) will be perpendicular to the sphere. Since the length of this vector is

$$d_n = \sqrt{(x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2} = r$$

we have that

$$\mathbf{u}_n = \left(\frac{x-x_c}{r}, \quad \frac{y-y_c}{r}, \quad \frac{z-z_c}{r} \right) \qquad \text{(LIGHT-16)}$$

at the point **P** = (x, y, z).

The formula, (LIGHT-16), is the only place where the specific geometric properties of the sphere enter into the calculations. The other three vectors, $\mathbf{u}_s$, $\mathbf{u}_r$, and $\mathbf{u}_o$, are calculated directly using formulas (LIGHT-3a-3c), (LIGHT-10), and (LIGHT-7a,b) as previously given.

Program code which will then implement the lighting model, (LIGHT-13), for the image of a sphere set up as indicated in Figure 7 would look something like the following (in the C programming language).

Assume that the following initial quantities have been assigned as floating point values:

IA = value of I_a, the ambient light intensity, formula (LIGHT-1)
IP = value of I_p, the point source light intensity, formulas (LIGHT-2, 5)
KA = value of the ambient light reflection coefficient, k_a, formula (LIGHT-1)
KD = value of the diffuse light reflection coefficient, k_d, formulas (LIGHT-2, 4)
KS = value of specular light reflection coefficient, k_s, formula (LIGHT-5)
XS, YS, ZS, the coordinates of the point source light
X0, Y0, Z0, the coordinates of the observer
XC, YC, ZC, the coordinates of the center of the sphere
R, the radius of the sphere
N, the exponent used in computing the specular reflection intensity

We will then use identifiers as follows:

```
float us[3];        // array to hold the three components of us
float un[3];        // array to hold the three components of un
float ur[3];        // array to hold the three components of ur
float uo[3];        // array to hold the three components of uo

float ds, dr, dob; // used in normalizing three of the vectors, uk
float x, y, z;              // coordinates of current point on the sphere
float zsq, dy, dx, rsq;     // storage of intermediate quantities
flost usun, uruo;  // storage for vector dot products
float Itot;                 // final total computed intensity
```

The block that implements (LIGHT-13) for this sphere example is then simply:

```
rsq = R^2;
for (x = xc – R; x <= xc + R; x+=1.0) {
        dx=x – xc;
```

```
        for (y = yc – R; y <= yc + R; y+=1.0)
        {
                dy = y – yc;
                zsq = rsq – dx*dx – dy*dy
                if (zsq >= 0.0)
                {                                       // point is on sphere image
                        z = sqrt(zsq);                  // (x, y, z) now known,
                        us[0] = XS - x; us[1] = YS – y; us[2] = ZS – z;
                        ds = sqrt(us[0]*us[0]+us[1]*us[1]+us[2]*us[2]);
                        for (j = 0; j < 3; j++) us[j] /= ds;
                        // us now complete

                        un[0] = (x-XC)/R; un[1] = (y-YC)/R; un[2] = (z-ZC)/R;
                        // un now complete

                        uo[0] = XO-x; uo[1] = YO-y; uno2] = ZO-z;
                        dob = sqrt(uo[0]*uo[0]+uo[1]*uo[1]+uo[2]*uo[2]);
                        for (j = 0; j < 3; j++) uo[j] /= dob;
                        // uo now complete

                        usun = us[0]*un[0]+us[1]*un[1]+us[2]*un[2];
                        for (j = 0; j < 3; j++) ur[j] =2.0*usun*un[j]-us[j];
                        dr = sqrt(ur[0]*ur[0]+ur[1]*ur[1]+ur[2]*ur[2]);
                        for (j = 0; j < 3; j++) ur[j] /= dr;
                        // ur now complete

                        //Now compute the total intensity: formula (LIGHT-13),
                        //(assume fatt = 1 here.)
                        uruo = ur[0]*uo[0]+ur[1]*uo[1]+ur[2]*uo[2];
                        Itot = IA*KA + IP*(KD*usun + KS*pow(uruo, N));

                        // Now need some way of converting Itot into a pixel “color”.
                        // then draw the pixel at (x, y) in that color.

        } // end of for (y = …)
} // end of for (x = …)
```

This is a very rough shot at coding (LIGHT-13) for an illuminated sphere, but it should give you an idea of how to go about implementing such an algorithm. Some repetitious code could be eliminated by developing a small function like Vdot(a, b), which would return the vector dot product of **a** and **b**. Near the end, pow() refers to a function in the standard C run-time library which will calculate general powers of a floating point number (prototype in <math.h>).

♦♦♦