# Chapter 6: Lighting

The last few years have seen an explosion in the use of computer graphics and realistic image synthesis. Realistic image synthesis is the process of creating synthetic images that are indistinguishable from images (photographs) of the real world. The rays of light or energy coming out of the light sources are bouncing around the room and finally reach the eye. To construct a picture of this room via computer graphics, we have to simulate this illumination process and be able to calculate the shading at each point of each surface in our scene. Calculating accurate values for the correct illumination of surfaces is possibly the most difficult task in computer graphics.

## *6.1 Light Sources*

We can thing of a light source as a surface emitting the light from every point *(x, y, z)* that is characterized by the direction of emission and the intensity of energy emitted at each wavelength λ as illustrated in Figure 6.1a. Thus, a general light source is six-dimensional *illumination function* $\mathbf{I}(x, y, z, \theta, \phi, \lambda)$. However, the model of human visual system is based on three-color theory, thus we can model the light source as having three components red, green, and blue and combine them to obtain the color that a human observer sees. Therefore, the light source is a vector $\mathbf{I} = (I_r, I_g, I_b)^T$ with independent red, green, and blue components. We will introduce four basic types of light sources: ambient lighting, point source, spotlight, and distant light.

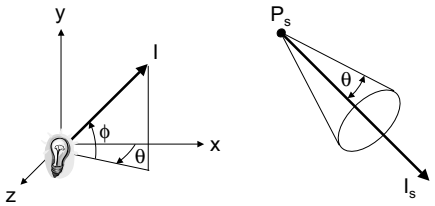





Figure 6.1. Light sources. Left) Illumination function. Right) Spotlight.

### 6.1.1 Ambient Light

The difficulties with accurately calculating interreflection between objects cause problems, and computer graphics community avoid them by approximating the total interreflection by a constant. In other words, we assume that there is a uniform light level, which illuminates each surface in the scene. This uniform lighting is called ambient light characterized by an intensity $\mathbf{I}_a = (I_{ar}, I_{ag}, I_{ab})^T$.
Global ambient term independent of any of the sources can be defined in OpenGL with the code

```
GLfloat global_amb[] = {0.2, 0.2, 0.2, 1.0};/* gray ambient light */
glLightModelfv(GL_LIGHT_AMBIENT, global_amb);
```

### 6.1.2 Point Light Source

An ideal point light source located at a position $p_0$ emits light equally in all directions with intensity $I(p_0)$. The intensity of illumination received from a point source is proportional the distance between the source and surface given by a *distance-attenuation function*

$$f(d) = \frac{1}{a + bd + cd^2},$$

where $d$ is the distance between the source and the surface. Hence, the received intensity by the surface at point $\mathbf{p}$ is $I(p, p_0) = f(d)\, I(p_0)$. The constant, linear, and quadratic terms characterized by constants $a$, $b$, and $c$ are set in OpenGL by

```
glLightf(GL_LIGHT0, GLCONSTANT_ATTENUATION, a);
```

**Example 6.1: Point Light Source in OpenGL.**

Each light source must have individually defined its position, the amount of ambient, diffuse, and specular light in homogeneous coordinates:

```
GLfloat light0_position[] = {2.0, 22.0, 66.0, 1.0};
GLfloat diffuse0[] = {0.0, 1.0, 0.0, 1.0};     /* green diffuse component */
GLfloat ambient0[] = {0.0, 1.0, 0.0, 1.0};     /* green ambient component */
GLfloat specular0[] = {1.0, 1.0, 1.0, 1.0};    /* white specular component */
```

Next we must enable both lighting and the particular source

```
glEnable{GL_LIGHTING};
glEnable{GL_LIGHT0};
glLightf(GL_LIGHT0, GL_POSITION, light0_position);
glLightf(GL_LIGHT0, GL_AMBIENT, ambient0);
glLightf(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightf(GL_LIGHT0, GL_SPECULAR, specular0);
```

### 6.1.3 Spotlight

Spotlights are narrow beams of light rays which can be modeled effectively by a cone whose apex is at light position $\mathbf{p}_s$, which points in direction $\mathbf{l}_s$, and whose width is determined by an angle $\theta$, as shown in Figure 6.1b. Note that for θ = 180°, the spotlight becomes a point source.

### 6.1.4 Distant Light

If the source is far from the surface we can replace a point source with a source that illuminates objects with parallel rays of light, a parallel source. The definition of parallel light in homogeneous coordinates is similar to the point source but the source location is replaced with the direction (with the fourth coordinate set to zero):

```
GLfloat light0_direction[] = {2.0, 2.0, 2.0, 0.0};
glLightf(GL_LIGHT0, GL_POSITION, light0_ direction);
```

## *6.2 Rendering equation*

Consider a simple scene with a single light source and two surfaces A and B. Some light that reaches the surface A is reflected to B where some of it is again reflected to A, and so on. This light transport is mathematically an integral equation, the *rendering equation*, which is used to find the shading of all surfaces in a scene. Because rendering equation cannot be solved even by numerical methods for a general scene, various simplifications and approximations, such as ray tracing and radiosity were discovered.
Let us try to calculate the amount of light energy that lives from $x$ in direction $\vec{\omega}$. The quantity that measures the light energy from $x$ in direction $\vec{\omega}$ is called radiance, $L_0(x,\vec{\omega})$ and is illustrated in Figure 6.2. Radiance is a five-dimensional quantity with three quantities for position and two for the direction $\vec{\omega} = (\theta, \phi)$. The direction, $\vec{\omega}$, is called the reflected direction. Point $x$ could be emitting light energy, when it is a light source, in which case, a certain energy $L_e(x,\vec{\omega})$ from $x$ is transported in direction $\vec{\omega}$.

There may be other light sources in the scene, and other reflecting surfaces. The total incoming light from these objects at $x$ is scattered by roughness of the surface and a certain amount $L_r(x,\vec{\omega})$ is reflected in direction $\vec{\omega}$. Thus we can write that outgoing radiance, $L_0$, is the sum of emitted radiance, $L_e$, and the reflected radiance, $L_r$:

$$L_0(x,\vec{\omega}) = L_e(x,\vec{\omega}) + L_r(x,\vec{\omega}).$$

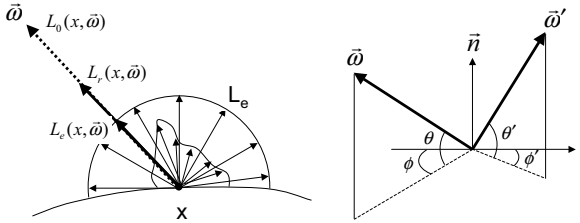





Figure 6.2. Emitted and reflected radiance.

Let us examine the reflected component $L_r$ further. We consider any other point $D$ in the scene and examine how the light energy coming from $D$ could be reflected in direction $\vec{\omega}$. If we note the incoming direction from $D$ as direction $\vec{\omega}' = (\theta', \phi')$ then the light energy contribution from $\vec{\omega}'$ to $\vec{\omega}$ can be written as

$$f_r(x,\vec{\omega}',\vec{\omega})L_i(x,\vec{\omega}')(\vec{\omega}' \cdot \vec{n}),$$

where $L_i(x,\vec{\omega}')$ is the incoming (or incident) radiance from direction $\vec{\omega}'$ to point $x$ and $f_r(x,\vec{\omega}',\vec{\omega})$ is fraction of energy arriving at $x$ which is reflected to direction $\vec{\omega}$. Here $\vec{n}$ is the surface normal at point $x$ and $\Omega$ is the hemisphere of incoming directions at $x$. Note that $\vec{\omega}' \cdot \vec{n} = \cos\theta'$. If we consider all possible incoming direction we see that the reflected radiance, $L_r$, is integral over the hemisphere and $d\vec{\omega}'$ is a solid angle. Substituting this to the outgoing radiance we find that:

$$L_0(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_\Omega f_r(x,\vec{\omega}',\vec{\omega})L_i(x,\vec{\omega}')(\vec{\omega}' \cdot \vec{n})d\vec{\omega}'.$$

This is the *rendering equation* used in ray-tracing algorithms.

## *6.3 Local Illumination Model*

In this section we will introduce tools for modeling the local light scattering at surfaces. We will investigate what happens when a beam of light strikes a given surface. In the graphics this is also known as *local illumination*.

### 6.3.1 The BRDF

The Bidirectional Reflectance Distribution Function, BRDF, was introduced as a tool for describing reflection of light at a surface. For the BRDF it is assumed that light striking a surface location is reflected at the same surface location. Thus BRDF is a seven-dimensional function having tree parameters for position, and two for each direction.

The BRDF, $f_r$, defines the relationship between reflected radiance and incident radiance:

$$f_r(x,\vec{\omega}',\vec{\omega}) = \frac{L_r(x,\vec{\omega})}{L_i(x,\vec{\omega}')(\vec{\omega}' \cdot \vec{n})d\vec{\omega}'}.$$

An important property of the BRDF is Hlemholtz's law of reciprocity, which states that it is independent of the direction in which light flows:

$$f_r(x,\vec{\omega}',\vec{\omega}) = f_r(x,\vec{\omega},\vec{\omega}').$$

Another important property of BRDF is due to the energy conservation. A surface cannot reflect more light than it receives:

$$\int_\Omega f_r(x,\vec{\omega}',\vec{\omega})L_i(x,\vec{\omega}')(\vec{\omega}' \cdot \vec{n})d\vec{\omega}' < 1, \ \forall\vec{\omega}.$$

### 6.3.2 Diffuse Reflection

A surface with diffuse reflection is characterized by light being reflected in all directions when it strikes the surface, as shown in Figure 6.3a. A special case of diffuse reflection is Lambertian or ideal reflection, in which the reflected direction is perfectly random, see Figure 6.3b. As a result the reflected radiance is constant in all directions regardless of the incident radiance. This gives the constant BRDF, $f_{r,d}(x)$ and then the light energy contribution from $\vec{\omega}'$ to $\vec{\omega}$ can be written as

$$f_{r,d}(x)L_i(x)(\vec{\omega}' \cdot \vec{n}).$$

We can simplify the above equation if we consider the reflection coefficient $k_d$ that represents the fraction of incoming diffuse light that is reflected. Note that the reflection coefficient corresponds to the constant BRDF. If $I_l$ is the intensity of the point light source then the diffuse reflection equation for a point on the surface can be written as

$$I_{l,diff} = k_d I_l(l \cdot \vec{n}),$$

where $l$ is the unit direction vector to the point light source from a surface position $x$. Note, that we obtain the same value for the illumination, regardless of how we observe the point.

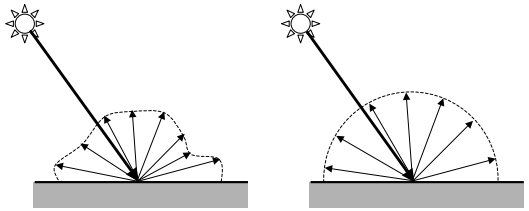

Figure 6.3. Diffuse material. Left) General diffuse reflection. Right) Lambertian diffuse reflection.

### 6.3.3 Specular Reflection

When we look at an illuminated shiny surface, typically a metallic surface, we see a highlight, or bright spot, at certain viewing directions. Most surfaces have some imperfections and as a result light is reflected in a small cone around the mirror direction as shown in Figure 6.4a. These surfaces are called glossy. For perfectly smooth surfaces we have the perfect specular reflection and the light is reflected only in the mirror direction, see Figure 6.4b. The mirror direction, $\vec{\omega}_s$, is

$$\vec{\omega}_s = 2(\vec{\omega}' \cdot \vec{n})\vec{n} - \vec{\omega}'.$$

We can express the perfect mirror reflection as a BRDF by using spherical coordinates for the directions;

$$f_r(x,\vec{\omega}',\vec{\omega}) = 2\rho_s\delta(\sin^2\theta' - \sin^2\theta)\delta(\phi' - \phi \pm \pi),$$

where Dirac's delta function, $\delta(x)$, is nonzero only when $x = 0$. Note that $\vec{\omega} = (\theta, \phi)$ and $\vec{\omega}' = (\theta', \phi')$.

**Fresnel reflection:** Fresnel reflection can be observed on metals and dielectrics. The reflected radiance due to specular reflection is

$$L_r(x,\vec{\omega}_s) = \rho_s(x)L_i(x,\vec{\omega}'),$$

where $\rho_s(x)$ is the fraction of the incident light that is reflected by the surface. If the reflected fraction of light is given by the Fresnel reflection coefficient we will obtain the Fresnel reflections. For unpolarized light we can use a good approximation of Fresnel reflection coefficient

$$\rho_s(x) = F_r(\theta) \approx F_0 + (1-F_0)(1-\cos\theta)^5,$$

where $F_0$ is the value of the real Fresnel reflection coefficient at normal incidence.

**Refraction:** For a ray of light in medium with index of refraction $\eta_1$ that strikes a transparent material with index of refraction $\eta_2$, we can compute the mount of light refracted into the object. The geometry of refraction is shown in Figure 6.5. Using the Snell's law, the direction, $\vec{\omega}_r$, of the refracted ray for a smooth surface with normal $\vec{n}$ is computed as:

$$\vec{\omega}_r = -\frac{\eta_1}{\eta_2}(\vec{\omega} - (\vec{\omega}\cdot\vec{n})\vec{n}) - \left(\sqrt{1-\left(\frac{\eta_1}{\eta_2}\right)^2(1-(\vec{\omega}\cdot\vec{n})^2)}\right)\vec{n}.$$

The amount of transmitted light for the refracted ray can be calculated as $\rho_s(x) = 1 - F_r$.

**The Phong specular model:** Phong proposed on approximate model that can be computed with only a slight increase over the work done for diffuse surfaces. The Phong specular model uses the equation

$$I_s = k_s L_s(\vec{h}\cdot\vec{n})^{n_s},\ \vec{h} = \frac{l+v}{|l+v|},$$

where $v$ is the unit viewing direction. The $k_s$ ($0 \le k_s \le 1$) is the fraction of the incoming specular light that is reflected and the exponent $n_s$ is a shininess coefficient. Values in range 100 to 500 correspond to most metallic surfaces, and infinity gives the mirror reflection.

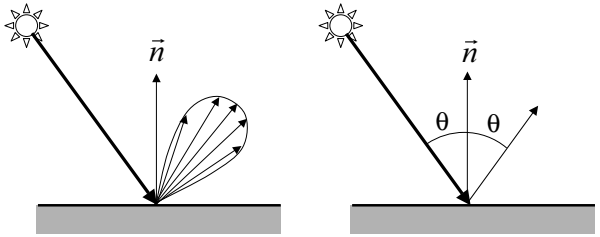


Figure 6.4. Specular surface. Left) Glossy specular reflection (i.e. a rough mirror). Right) Perfect specular reflection.

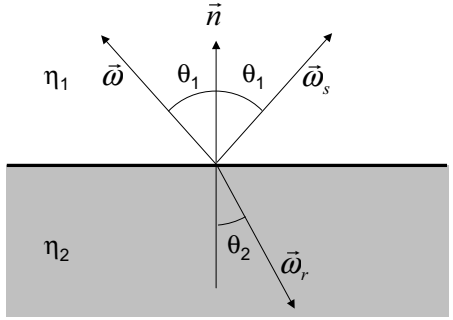


Figure 6.5. The geometry of reflection and refraction.

### 6.3.4 Putting the Ambient, Diffuse and Specular Components Together

The Phong reflection model is a phenomological model with no physical basis. However, it is simple and it is implemented in OpenGL.

For a single point light source, we can combine the ambient, diffuse and specular reflections from a point on a surface. The contribution of each component is controlled by ambient, $k_a$, diffuse, $k_d$, and specular, $k_s$, coefficients. Therefore the Phong model is written as:

$$I = I_{ambient} + I_{diff} + I_{spec} = k_a I_a + k_d I_l(l\cdot\vec{n}) + k_s I_l(\vec{h}\cdot\vec{n})^{n_s}.$$

Figure 6.6. illustrates the results of illumination with ambient light for different constants $k_a$ in top row, the bottom row illustrates the specular reflection for different constants $k_s$. For an RGB color representation we can set surface color by specifying the ambient, diffuse, and specular coefficients as three-element vectors. The diffuse vector, for example, would then have RGB components ($k_{dR}$, $k_{dG}$, $k_{dB}$). The intensity calculation for multiple light sources, $I_{li}$, and distance attenuation function, $f_i(d)$, assigned for each light source can be expressed for each color component separately. To use multiple light sources in a scene we must sum the contributions from individual light sources. We write the Phong model for a B color component as

$$I_B = k_{aB}I_{aB} + \sum_{i=1}^{n} f_i(d)I_{lBi}\left[k_{dB}(l_i\cdot\vec{n}) + k_{sB}(\vec{h}_i\cdot\vec{n})^{n_s}\right]$$

OpenGl uses three-element vectors to set surface color, for example, we might define ambient, diffuse, and specular coefficients ($k_a$, $k_d$, $k_s$) for each primary color RGB through arrays:

```
GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat diffuse[] = {1.0, 0.8, 0.0, 1.0};
GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);
```

The shiness exponent in the specular reflection term is specified as

```
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100.0);
```

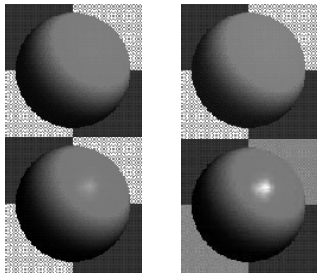

Figure 6.6. Top row: Sphere illuminated with different ambient light. Bottom row: Result of increasing the specular parameters.

## *6.4 Global Illumination Model*

The local illumination model has limitations. For example all objects are shaded separately not taking into account inter-reflections with other objects in the scene. It is the responsibility of the global illumination algorithm to calculate how light leaving the light source interacts with the scene.

### 6.4.1 Neumann Series Solution to Rendering Equation

If we look at the rendering equation closely we see that the radiance is on both sides of the equation, which prevents us to solve the equation directly. Consider again our simple scene with a single light source and two surfaces A and B. Some light that reaches the surface A is reflected to B and again reflected to A. In such case the quantity that we wish to calculate will appear on both sides of equation and thus makes it very difficult to solve. One way to get around this problem is to recursively replace the incident radiance on the right side with the expression for the radiance. It is more convenient to write the rendering equation expressed in terms of an integral operator, *K*:

$$L_0 = L_e + KL_0,$$

Here $L_0$ corresponds to the radiance term, $L_e$ corresponds the emitted radiance, and the integral operator *K* applied on a function *g* is defined by:

$$Kg = \int_{\Omega} f_r(x,\vec{\omega}',\vec{\omega})g(x,\vec{\omega}')(\vec{\omega}'\cdot\vec{n})d\vec{\omega}'.$$

Thus, by recursive evaluation the solution of the rendering equation is the Neumann series:

$$L_0 = L_e + KL_e + K^2L_e + K^3L_e + ... = \sum_{m=0}^{\infty} K^m L_e.$$

Each application of the operator, *K*, corresponds to an additional light-surface interaction along a ray path from light source to the eye. The Neuman series is the basis for several ray-tracing algorithms.

### 6.4.2 Ray Tracing

*Ray tracing* is an extension of the local illumination approach. In ray tracing method the light rays are processed in inverse direction. It is assumed that all rays start at the eye of an observer (the center of projection), pass through the image plane toward objects in a scene, and finally reach the light sources. This approach reduces the number of cast rays. Knowing that we must assign a color to every pixel, we must cast at least one ray through each pixel and compute the color (the average radiance).

**Definition 6.1:** A ray, *r*, has the form

$$r(x,\vec{\omega}) = x + d\vec{\omega},$$

where *x* is the origin of the ray, $\vec{\omega}$ is the direction of the ray, and *d* is the distance moved along the ray. The rays from observer through the pixels are called the *primary rays*.

To compute the radiance we must find the nearest object intersected by the ray (smallest *d*). For every intersection point reflected ray in direction $\vec{\omega}_s$ and refracted ray in direction $\vec{\omega}_r$ are generated. These rays can hit the other objects along their path as illustrated in Figure 6.7a. Therefore, ray tracing is perceived as a binary tree parsing process. Nodes of the *ray tree* are intersection points and edges are the rays as shown in Figure 6.7b. The recursive nature of the algorithm can be used in implementation by the recursive calls of a ray tracing procedure for every node of the ray tree.

Given an intersection point, *p*, we need to find the outgoing radiance, $L_0$, in the direction of the ray. The local illumination models are used to calculate the radiance at *p*, for example the Phong model (used by OpenGL) or the rendering equation. For this purpose we need to know the surface normal, $\vec{n}$, at *p* as well as the BRDF, $f_r$ or the specular coefficients $k_a$, $k_d$, $k_s$. With this information we can compute the illumination from each light source by estimating the incident radiance at *p*.

Finally, the color of a pixel is the sum of all local radiances calculated for every intersection point of a primary ray along its path. The radiance is summed up starting from leaves of the tree toward its root.

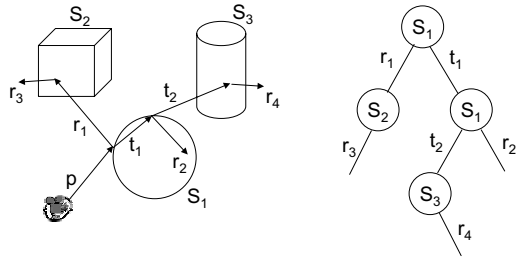


Figure 6.7. Recursive ray tracing. Left) Reflected and refracted ray paths through a scene. Right) Binary ray tree for the paths shown. Nodes are the intersected points of objects and edges are rays.

Unfortunately, ray tracing is not a full global illumination process. It cannot compute the indirect illumination on the diffuse surfaces. In the light transport notation using the regular expression we find that ray tracing can compute light path of the form:

LD?S*E

where L is the light source, E is the eye, S is a specular reflection, and D is a diffuse reflection. Any particular type of ray path can be represented by a string of symbols from the alphabet {L, D, S, E}. The symbol '*' indicates any number of repetitions of term, and the symbol '?' indicates zero or one repetition. For example S* is a set {0, S, SS, SSS, …} indicating the infinite number of ray bounces via specular reflection. Notice that the ray tracing cannot handle the ray paths containing the sequence DSD or DD. We can conclude that ray tracing works well for highly specular surfaces and translucent objects, such as glass.

### 6.5.3 Radiosity

If we make the assumption that all surfaces are perfectly diffuse, we can simplify the rendering equation such that the numerical solution can be found by a *radiosity* method. Radiosity is best suited for scenes with perfectly diffuse surfaces, such as interiors of buildings.

The basic method brakes up the scene into small flat polygons, or patches, each of which can be assumed to be perfectly diffuse and renders in constant shade. The problem is to find these shades. We can find them in two steps. In the first step, we consider patches pairwise to determine form factors that describe how the light energy leaving one patch affects the other. Once the form factors are determined, the rendering equation can be reduced to a set of linear equations for the radiosities of patches. Once we solve these equations, we can render the scene using any renderer with flat shading.

**Form factor** $F_{jk}$ is the fractional amount of outgoing radiance from surface $j$ that reaches surface $k$. Note that $F_{kk} = 0$. A form factor is a dimensionless quantity. The accurate calculation of the form factor, $F_{jk}$, involves placing a hemisphere, with unit radius, at a point on a surface $j$. The second surface, $k$, is projected on to the hemisphere, and then projected onto the base of the hemisphere. The form factor is then the area projected on the base of the hemisphere divided by the area of the base of the hemisphere, as illustrated in Figure 6.8.

**Radiosity equation** is derived from rendering equation assuming $N$ diffuse surfaces in the scene. The outgoing energy is expressed with

$$B_k = E_k + \rho_k \sum_{j=1}^{N} B_j F_{jk},$$

where radiosity parameter $B_k$ is the total rate of energy leaving the surface $k$ per unit area, $E_k$ is the emitted radiance (nonzero for light sources), parameter $\rho_k$ is the reflectivity factor for surface $k$. For a given $E_k$, $\rho_k$, and $F_{jk}$ we must find unknown $B_k$. We can rewrite the radiance equation into the system of linear equations with unknowns on the left side

$$(1-\rho_k F_{kk})B_k - \rho_k \sum_{j \neq k} B_k F_{jk} = E_k, \quad k = 1,2,3,...,N.$$

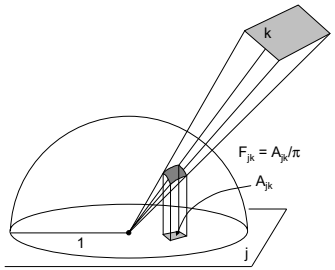


Figure 6.8. Form factor calculation.

### 6.5.4 Stochastic Methods

Photon mapping is an efficient global illumination technique for realistic image synthesis that has been developed in computer graphics in the last few years. With photon mapping it is easy to simulate caustics (for example, the light focused through a glass onto a table), color bleeding (such as the soft rendering of white wall due to an adjacent red carpet), participating media (for example, a room filled with smoke), and subsurface scattering (noticeable for example on plant leaves).

The first step is building a photon map using photon tracing and counting the photons. This photon can then be visualized directly by using a simplified ray tracer. This ray tracer uses the radiance estimate from the photon map to compute the reflected radiance from all diffuse materials and standard recursive ray tracing for specular materials. The paths traced by photons are L(S|D)*D and the paths traced by the ray tracer are (LS*E)|(DS*E), where symbol '|' indicates the logical OR operation. The combination of those paths shows that the method does indeed trace all paths between the eye and the light source.

## *6.6 Program: Lighting with OpenGL*

## *Exercises*

6.1 Show that the halfway vector $\vec{h}$ is at the angle at which a surface must be oriented so that the maximum amount of reflected light reaches the viewer.

6.2 Write a pseudocode routine ray that recursively traces a cast ray. You can assume that you have a function available that will intersect a ray with an object. Consider how to limit how far the original ray will be traced.

6.3 Write a program in OpenGl to use Phong illumination method using a single point light source, ambient light, and specular surfaces. Try to create light sources that move in the scene.

6.4 Calculate the intersection point of a ray with sphere and ray with polygon.

6.5 Discuss the acceleration techniques for ray intersection with objects in a scene.

6.6 Show that the halfway vector $\vec{h}$ is in the same plane as $v$, $l$, and $\vec{n}$.

6.7 Proof the equations for reflected direction, $\vec{\omega}_s$, and refracted direction, $\vec{\omega}_r$ using Snell's law $\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$.

## *References*


Angel A. 2000. Interactive Computer Graphics a Top Down Approach with OpenGL, Addison-Wesley, Reading, Massachusetts, 2nd edition.

Hearn D. and Baker M.P. 1994. Computer Graphics, Prentice Hall, New Jersey, 2nd edition.

Jensen H. W. 2001. Realistic Image Synthesis Using Photon Mapping, A K Peters, Natick, Massachusetts.

Cohen M.F. and Wallace J.R. Radiosity and Realistic Image Synthesis, …