MONTE-CARLO METHODS IN GLOBAL ILLUMINATION

Script written by
Szirmay-Kalos László
in WS of 1999/2000

Institute of Computer Graphics
Vienna University of Technology

Contents

Chapter 1

Introduction

The ultimate objective of **image synthesis** or **rendering** is to provide the user with the illusion of watching real objects on the computer screen (figure 1.1). The image is generated from an internal model which is called the **virtual world**. To provide the illusion of watching the real world, the color sensation of an observer looking at the artificial image generated by the graphics system must be approximately equivalent to the color perception which would be obtained in the real world. The color perception of humans depends on the **light power** reaching the eye from a given direction and on the operation of the eye. The power, in turn, is determined from the **radiance** of the visible points. The radiance depends on the shape and optical properties of the objects and on the intensity of the lightsources. In order to model this complex phenomenon, both the physical-mathematical structure of the light-object interaction and the operation of the eye must be understood.

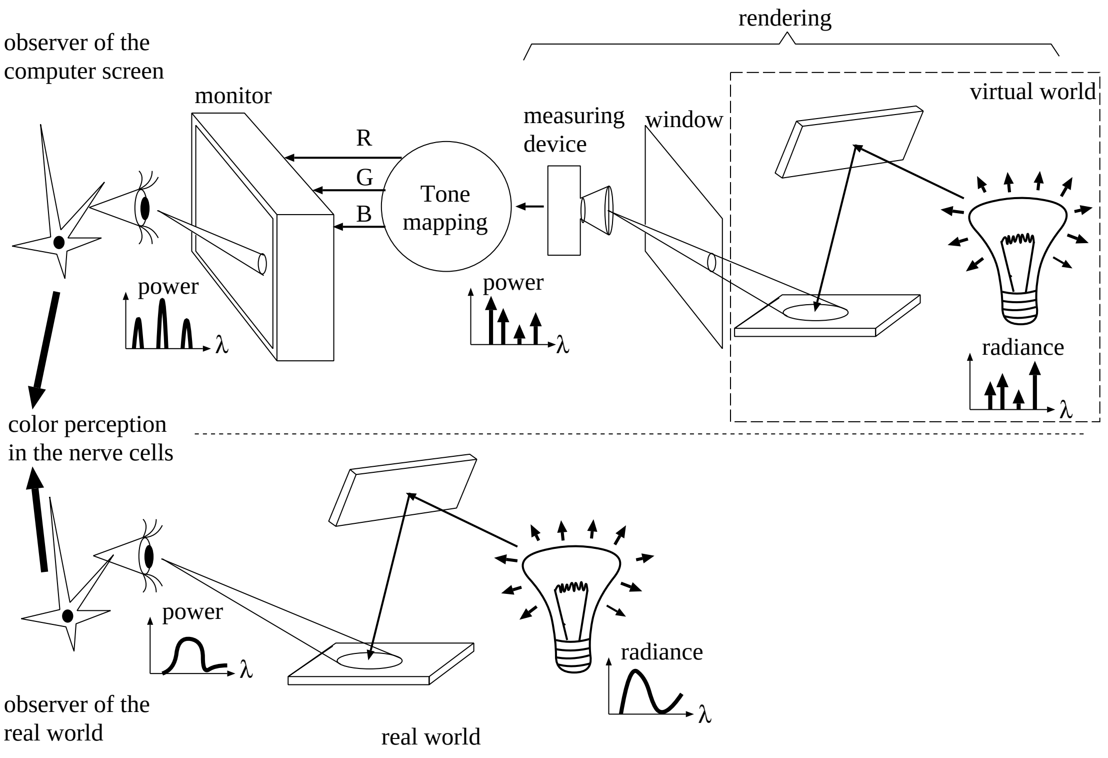


Figure 1.1: Tasks of rendering

The image synthesis uses an internal model consisting of the **geometry of the virtual world**, **optical material properties** and the description of the **lighting** in the scene (figure 1.2). From these, applying the laws of physics (e.g. Maxwell equations) the real world optical phenomena can be simulated to find the light distribution in the scene. This step is called the view-independent step or the **global pass** of rendering. Then a **measurement device**, called the **eye** or **camera**, is placed into the scene and the light distribution is measured from a given location and orientation. This is called the view-dependent step or the **local pass**. Note that not all rendering algorithms make a clear distinction between the determination of the view-independent light distribution and the measurement of this distribution by the camera, but simultaneously compute the light distribution and its effect on the camera.

Rendering results in a representation of the perceived image, which is usually the collection of pixel colors or some discrete sampling of the radiance function. The exact simulation of the light perceived by the eye is impossible, since it would require endless computational process. On the other hand, it is not even worth doing since the possible distributions which can be produced by computer screens are limited in contrast to the infinite variety of real world light distributions. Consequently, color perception is approximated instead of having a completely accurate simulation. The accuracy of this approximation is determined by the ability of the eye to make a distinction between two light distributions.

Computer screens can produce controllable electromagnetic waves, or colored light, mixed from three separate wavelengths for their observers. Thus in the final step of image synthesis **tone mapping** is needed which converts the computed color or radiance information to the R, G, B intensities that can be produced by the color monitor.

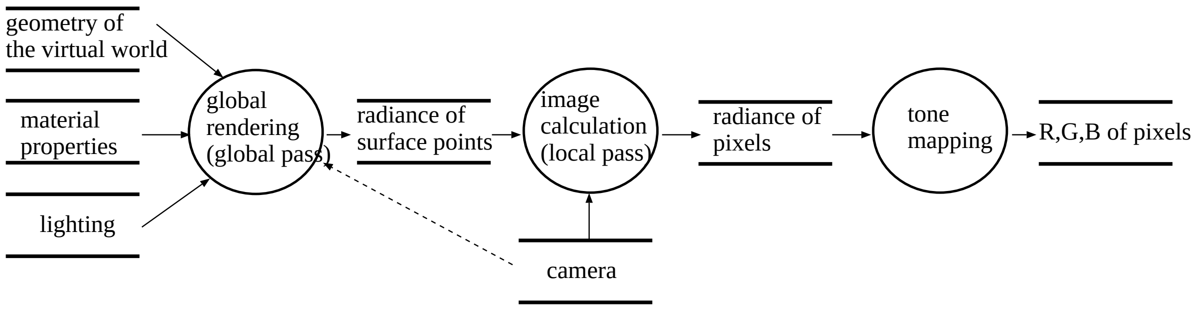

geometry of the virtual world
material properties
lighting
global rendering (global pass)
radiance of surface points
image calculation (local pass)
radiance of pixels
tone mapping
R,G,B of pixels
camera

Figure 1.2: Dataflow of rendering

1.1 Global pass

The **global pass** determines the light reflected off the surface points at different directions. Since light is an electromagnetic wave, light distribution in a point and at a given direction can be represented by a wavelength-dependent function [Ábr97, Kón85]. Rendering algorithms usually evaluate this functions at a few representative wavelengths. On a given wavelength the intensity of the light is described by the **radiance**. In scenes not incorporating **participating media** it is enough to calculate the radiance at surface points. The radiance reflected off a surface point is affected by the emission of this point (**lighting**), the illumination provided by other surface points and the optical properties of the material at this point (**material properties**). Formally this dependence is characterized by a Fredholm type integral equation of the second kind, which is called the **rendering equation**. From mathematical point of view, global pass is the solution of this integral equation for the representative wavelengths.

1.2 Local pass

The **local pass** means the measurement of the global radiance function by a camera. A camera is a collection of light measuring devices which usually correspond to pixels in the image. A certain measuring device is characterized by a sensitivity function that describes which points and directions may affect the device.

1.3 Tone mapping

Light is an electromagnetic wave, and its **color** is determined by the eye's perception of its spectral energy distribution. Due to its internal structure, the eye is a very poor spectrometer since it actually samples and integrates the energy in three overlapping frequency ranges by three types of photopigments according to a widely accepted (but also argued) model. As a consequence of this, any color perception can be represented by three scalars (called **tristimulus** values) instead of complete functions [Ábr97, Kón85, Nem90].

A convenient way to define the axes of a coordinate system in the three-dimensional space of color sensations is to select three wavelengths where one type of photopigments is significantly more sensitive than the other two [SK99c]. This method has been devised by Grassmann, who also specified a criterion for separating the three representative wavelengths. **Grassmann laws** state that the representative wavelengths should be selected such that no one of them can be matched by the mixture of the other two in terms of color sensation (this criterion is similar to the concept of linear independence.) An appropriate collection of the representative wavelengths is:

$$\lambda_{\mathrm{red}} = 645 \text{ nm}, \quad \lambda_{\mathrm{green}} = 526 \text{ nm}, \quad \lambda_{\mathrm{blue}} = 444 \text{ nm}. \tag{1.1}$$

Now let us suppose that monochromatic light of wavelength λ is perceived by the eye. The equivalent portions of red, green and blue light, or (r, g, b) tristimulus values, can be generated by three **color matching functions** ($r(\lambda)$, $g(\lambda)$ and $b(\lambda)$) which are based on physiological measurements. Note the negative section of $r(\lambda)$ (and to a less extent in $g(\lambda)$) in figure 1.3. It means that not all colors can be represented by positive (r, g, b) values.

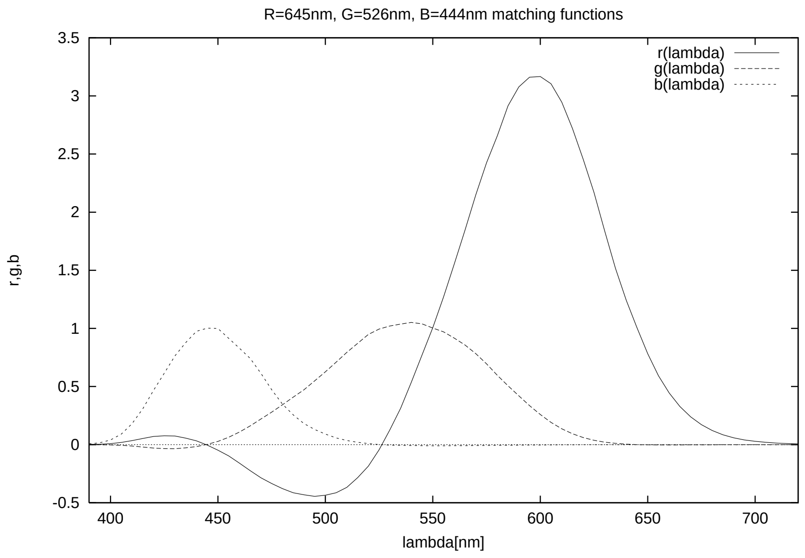


Figure 1.3: Mean 10-deg color matching functions of Stiles and Burch: $r(\lambda)$, $g(\lambda)$, $b(\lambda)$.

If the perceived color is not monochromatic, but is described by an $L(\lambda)$ distribution, the tristimulus coordinates are computed using the assumption that the sensation is produced by an additive mixture of the perceptions of elemental monochromatic components:

$$r = \int\limits_{\lambda} L(\lambda) \cdot r(\lambda)\, d\lambda, \quad g = \int\limits_{\lambda} L(\lambda) \cdot g(\lambda)\, d\lambda, \quad b = \int\limits_{\lambda} L(\lambda) \cdot b(\lambda)\, d\lambda. \tag{1.2}$$

For computer generated images, the color sensation of an observer watching a virtual world on the screen must be approximately equivalent to the color sensation obtained in the real world. Since color sensations are represented by (r, g, b), it means that the tristimulus values should be identical. If two energy distributions are associated with the same tristimulus coordinates, they produce the same color sensation, and are called **metamers**.

In computer monitors and in television screens three phosphor layers can be stimulated to produce red, green and blue light. The objective, then, is to find the necessary stimulus to produce a metamer of the real energy distribution of the light [Sch96, BS95]. This stimulus can be controlled by the (R, G, B) values of the actual pixel. The (r, g, b) matching functions of figure 1.3 depend on the wavelength of the selected primaries, which are not necessarily identical to the wavelengths on which our monitor can emit light. This requires the conversion of tristimulus values by a linear transformation.

The calculation of pixel R, G, B values thus consists of the following steps. First the spectrum associated with the pixel is computed. Then the spectrum is matched by three standard color matching functions defined by three primaries. Finally, the standard color coordinates are transformed to the monitor color coordinates taking into account the monitor properties. In practice, the standard color system is usually the **CIE XYZ** system [WS82] which uses three hypothetical primaries to allow the definition of any color by positive weights.

The linear transformation that converts from the **XYZ** system to the monitor **RGB** system can be obtained from the X, Y, Z coordinates of the emissions of the three phosphors and of the white point of the monitor. For a monitor with **standard NTSC phosphors** and **white point**, the following transformation can be used [Gla95]:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.967 & -0.548 & -0.297 \\ -0.955 & 1.938 & -0.027 \\ 0.064 & -0.130 & 0.982 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}. \tag{1.3}$$

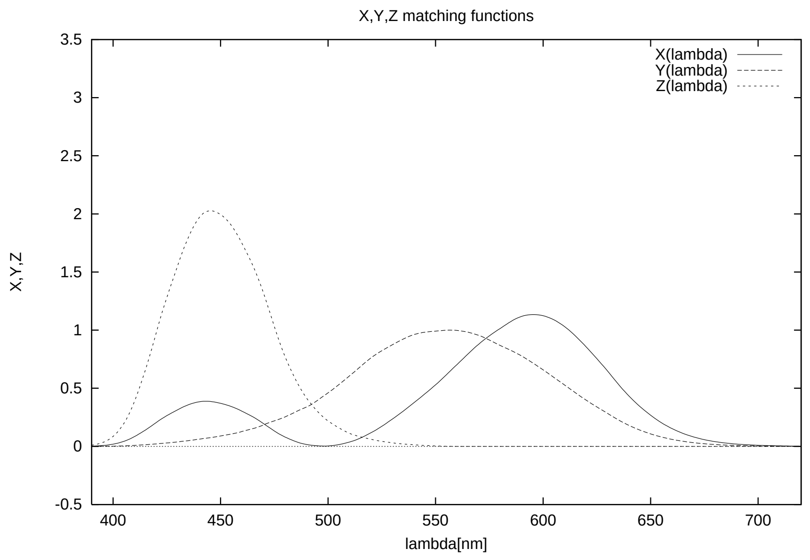


Figure 1.4: Mean 10-deg color XYZ matching functions of Stiles and Burch: $X(\lambda)$, $Y(\lambda)$, $Z(\lambda)$

The whole computation of the (R, G, B) values in order for the monitor color to be a metamer of the calculated spectrum is called **tone mapping**. The (R, G, B) values are positive numbers usually in the range of [0...255] if 8 bits are available to represent them. Unfortunately, not all colors can be reproduced on the computer screen, because of the negative sections of the color matching functions and due to the fact that the number of available intensity levels is usually much less than can be perceived in the real world. Thus tone mapping is also responsible for optimally selecting from the available intensity levels for color reproduction. The mapping from the computed levels to the available ones can be either linear or logarithmic. The latter takes advantage of the logarithmic characteristics of the human perception system [PP98].

Chapter 2

Global illumination problem

In this chapter the mathematical model of the light-surface interaction is presented. This mathematical model is an integral equation, which has to be solved to obtain physically accurate images.

2.1 The rendering equation

Hereinafter, monochromatic light of a representative wavelength λ will be assumed, since the complete color calculation can be broken down to these representative wavelengths. The parameters of the equations usually depend on the wavelength, but for notational simplicity, we do not always include the λ variable in them.

In this section, we briefly review the measures of the light transport and the mathematical formulation that can compute them.

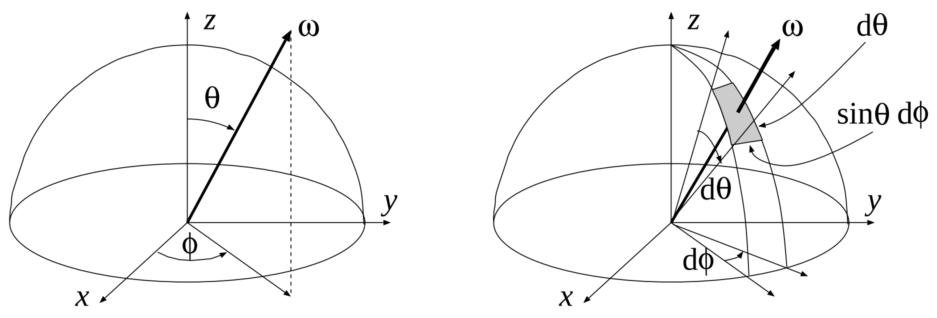


Figure 2.1: Definition of directions in a spherical coordinate system (left) and calculation of differential solid angles (right)

The directional property of the light energy emission is described in a so-called **illumination sphere** Ω or in **illumination hemisphere** Ω_H which contain those solid angles to where the surface point can emit energy. The surface of transparent materials can emit in any directions of a sphere, while the surface of opaque materials can transfer energy only to the hemisphere that is "above" the surface.

Setting up a **spherical coordinate system** (figure 2.1), a direction ω can be defined by two angles θ, ϕ, where θ is the angle between the given direction and the z-axis, and ϕ is the angle between the projection of the given direction onto the x, y plane and the x-axis.

Sets of directions are defined by solid angles. By definition, a **solid angle** is a cone or a pyramid, with its size determined by its subtended area of a unit sphere centered around the apex (figure 2.2). A differential (infinitesimal) solid angle can also be given by a vector $d\vec{\omega}$, where the vector equals to a direction of the differential set.

A **differential solid angle** can also be expressed by the θ, ϕ angles. Suppose that θ is modified by $d\theta$ and ϕ is by $d\phi$. During this the directional vector scans a differential rectangle having $d\theta$ vertical and $\sin\theta \cdot d\phi$ horizontal sizes (right of figure 2.1), thus the size of the solid angle is

$$d\omega = \sin\theta \cdot d\phi d\theta. \tag{2.1}$$

The solid angle, in which a differential dA surface can be seen from point $\vec{p}$, is the projected (visible) area per the square of the distance of the surface (figure 2.2). If the angle between the surface normal of dA and the

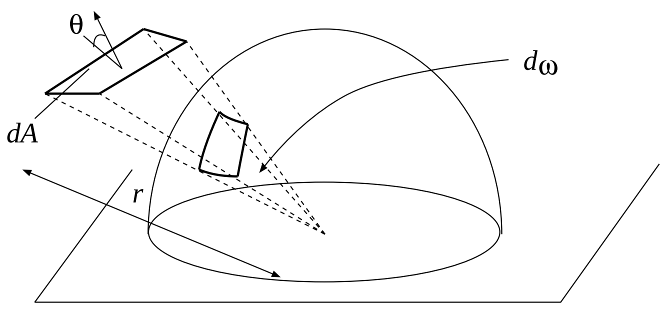


Figure 2.2: Definition of the solid angle

directional vector from dA to $\vec{p}$ is θ, and the distance from dA to $\vec{p}$ is r, then this solid angle is:

$$d\omega = \frac{dA \cdot \cos\theta}{r^2}. \tag{2.2}$$

The intensity of the energy transfer is characterized by several metrics in computer graphics depending on whether or not the directional and positional properties are taken into account.

The light **power** or **flux** Φ is the energy radiated through a boundary per unit time over a given range of the spectrum (say $[\lambda, \lambda + d\lambda]$). Since a **photon** has $\hbar/\lambda$ energy where $\hbar$ is the Planck-constant, the power can be supposed to be propotional to the number of photons that go through the boundary in a unit time. The power is not always a convenient measure since it also needs the definition of a boundary. We can get rid of this additional information if the boundary is defined in a differential way focusing on a single surface point and a single direction. The resulting measure is called the radiance.

The **radiance** or **intensity** $L(\vec{x}, \omega)$ is the differential light flux $\Phi(\vec{x}, dA, \omega, d\omega)$ leaving a surface element dA around $\vec{x}$ in a differential solid angle $d\omega$ around ω per the projected area of the surface element dA and the size of the solid angle $d\omega$. If the angle of the surface normal and the direction of interest is θ, then the projected area is $dA \cdot \cos\theta$, hence the radiance is:

$$L(\vec{x}, \omega) = \frac{d\Phi(\vec{x}, dA, \omega, d\omega)}{dA \cdot d\omega \cdot \cos\theta}. \tag{2.3}$$

Since a differential surface area can also be seen as a vector $d\vec{A}$, which is parallel to the surface normal at the given point, the radiance can also be obtained in a simpler form

$$L(\vec{x}, \omega) = \frac{d\Phi(\vec{x}, dA, \omega, d\omega)}{d\vec{A} \cdot d\vec{\omega}}. \tag{2.4}$$

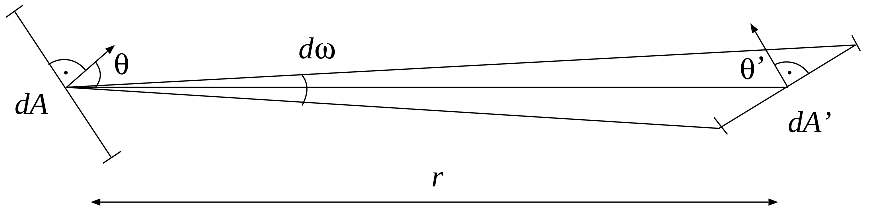


Figure 2.3: Energy transfer between two differential surface elements

Having introduced the most important metrics, we turn to their determination in the simplest case, where there are only two differential surface elements in the 3D space, one (dA) emits light energy and the other (dA') absorbs it (figure 2.3). According to the definition of the radiance (equation (2.3)), if dA' is visible from dA in solid angle $d\omega$ and the radiance of the surface element dA is L in this direction, then the flux leaving dA and reaching dA' is:

$$d\Phi = L \cdot dA \cdot d\omega \cdot \cos\theta. \tag{2.5}$$

Using equation (2.2), the solid angle can be expressed from the projected area of dA', thus we get:

$$d\Phi = L \cdot \frac{dA \cdot \cos\theta \cdot dA' \cdot \cos\theta'}{r^2}. \tag{2.6}$$

This formula is called the **fundamental law of photometry**.

Note that if equation (2.2) is used again for the emitter, the transferred power can also be written in the following form:

$$d\Phi = L \cdot \frac{dA \cdot \cos\theta \cdot dA' \cdot \cos\theta'}{r^2} = L \cdot dA' \cdot d\omega' \cdot \cos\theta'. \tag{2.7}$$

Thus similar formula applies for the patch that receives the power as for the patch that emits it.

In light-surface interaction the surface illuminated by an incident beam may reflect a portion of the incoming energy in various directions or it may absorb the rest. It has to be emphasized that a physically correct model must maintain energy equilibrium, that is, the reflected and the transmitted (or absorbed) energy must be equal to the incident energy.

Optically perfect or smooth surfaces will reflect or transmit only **coherent component**s governed by the laws of geometric optics, including the law of reflection and the Snellius–Descartes law of refraction. The surface irregularities, however, reflect or refract the incident light incoherently in any direction. Since the exact nature of these irregularities is not known, light-surface interaction is modeled by means of probability theory.

Assume that a photon comes from the direction denoted by ω' to point $\vec{x}$. The probability of reflection or refraction at $\vec{x}$ into solid angle $d\omega$ around ω is expressed by the following probability density function, also called as **transfer probability density**:

$$w(\omega', \vec{x}, \omega) \cdot d\omega = \Pr\{\text{photon is reflected or refracted to } d\omega \text{ around } \omega \mid \text{coming from } \omega'\}. \tag{2.8}$$

Note that this probability distribution is a mixed, discrete-continuous distribution, since the probability that the light is reflected exactly to the ideal reflection direction may be non-zero.

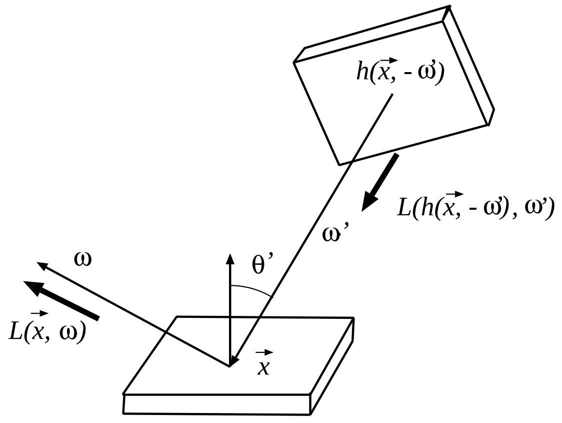


Figure 2.4: Geometry of the rendering equation

The light flux ($\Phi^{\rm out}$) leaving the surface at solid angle $d\omega$ around ω consists of the emission and reflected (or refracted) components.

In order to obtain the **reflected/refracted component**, let us assume that photons are coming to area dA around $\vec{x}$ from solid angle $d\omega'$ around ω' and their total represented power is $\Phi^{\rm in}(\vec{x}, dA, \omega', d\omega')$. The probability that a photon is reflected to $d\omega$ is $w(\omega', \vec{x}, \omega) \cdot d\omega$ thus the expected power leaving the surface element after reflection or refration is:

$$w(\omega', \vec{x}, \omega)\, d\omega \cdot \Phi^{\rm in}(\vec{x}, \omega', d\omega').$$

To obtain the total reflected/refracted power, all the possible incoming directions ω' should be taken into account and their contributions should be integrated:

$$\int_{\Omega} (w(\omega', \vec{x}, \omega)\, d\omega) \Phi^{\rm in}(\vec{x}, \omega', d\omega'). \tag{2.9}$$

If the surface itself emits energy, i.e. if it is a lightsource, then the **emission** also contributes to the output flux:

$$\Phi^e(\vec{x}, \omega) = L^e(\vec{x}, \omega) \cdot dA \cdot \cos\theta \cdot d\omega. \tag{2.10}$$

Adding the possible contributions we obtain:

$$\Phi^{\rm out}(\vec{x}, \omega) = \Phi^e(\vec{x}, \omega) + \int_{\Omega} (w(\omega', \vec{x}, \omega)\, d\omega) \Phi^{\rm in}(\vec{x}, \omega', d\omega'). \tag{2.11}$$

The fluxes can be expressed by the radiant intensities according to equations (2.5) and (2.7), thus:

$$\begin{aligned} \Phi^{\text{in}}(\vec{x}, \omega', d\omega') &= L^{\text{in}}(\vec{x}, \omega') \cdot dA \cdot \cos\theta' \cdot d\omega', \\ \Phi^{\text{out}}(\vec{x}, \omega, d\omega) &= L(\vec{x}, \omega) \cdot dA \cdot \cos\theta \cdot d\omega. \end{aligned} \tag{2.12}$$

Substituting these terms into equation 2.11 and dividing both sides by $dA \cdot d\omega \cdot \cos\theta$ we get:

$$L(\vec{x}, \omega) = L^e(\vec{x}, \omega) + \int_{\Omega} L^{\text{in}}(\vec{x}, \omega') \cdot \cos\theta' \cdot \frac{w(\omega', \vec{x}, \omega)}{\cos\theta}\, d\omega'. \tag{2.13}$$

Let us realize that $L^{\text{in}}(\vec{x}, \omega')$ is equal to $L(\vec{y}, \omega')$ if $\vec{y}$ is the point that is visible from $\vec{x}$ at direction $-\omega'$, usually expressed by the so called **visibility function** h (i.e. $\vec{y} = h(\vec{x}, -\omega')$).

Using these equations and introducing the **bi-directional reflection/refraction function** — or **BRDF** for short — as defined by the following formula

$$f_r(\omega', \vec{x}, \omega) = \frac{w(\omega', \vec{x}, \omega)}{\cos\theta}, \tag{2.14}$$

we obtain the following fundamental law, called the **rendering equation**:

$$L(\vec{x}, \omega) = L^e(\vec{x}, \omega) + \int_{\Omega} L(h(\vec{x}, -\omega'), \omega') \cdot f_r(\omega', \vec{x}, \omega) \cdot \cos\theta'\, d\omega'. \tag{2.15}$$

Introducing the following notation for the integral operator

$$(\mathcal{T}L)(\vec{x}, \omega) = \int_{\Omega} L(h(\vec{x}, -\omega'), \omega') \cdot f_r(\omega', \vec{x}, \omega) \cdot \cos\theta'\, d\omega', \tag{2.16}$$

the short form of the rendering equation can also be established:

$$L = L^e + \mathcal{T}L. \tag{2.17}$$

In fact, every color calculation problem consists of several rendering equations, one for each representative wavelength. Optical parameters (L^e, f_r) obviously vary in the different equations, but the geometry represented by function h does not.

2.2 Measuring the radiance

Having solved the rendering equation, the radiance can be determined for any point and direction. To obtain an image, the power that affect different parts of light sensitive material (retina or the film of a camera) must be determined. Mathematically each distinct part is associated with a measuring device, thus the collection of these devices is, in fact, the model of the human eye or cameras.

A measuring device is characterized by a **sensitivity function** $W^e(\vec{y}, \omega')$, which gives a **scaling value** C if a light-beam of unit power leaving $\vec{y}$ in direction ω' contributes to the measured value and 0 otherwise (for example, this device can measure the power going through a single pixel of the image and landing at the eye, or leaving a surface element at any direction).

Since the power leaving surface area $d\vec{y}$ around $\vec{y}$ in solid angle $d\omega$ around ω is $L(\vec{y}, \omega) d\vec{y} \cos\theta d\omega$, the measured value by a certain device is:

$$\int_{\Omega}\int_{S} d\Phi(\vec{y}, \omega) \cdot W^e(\vec{y}, \omega) = \int_{\Omega}\int_{S} L(\vec{y}, \omega) \cos\theta \cdot W^e(\vec{y}, \omega)\, d\vec{y}\, d\omega = \mathcal{M}L. \tag{2.18}$$

Operator $\mathcal{M}$ is called the **radiance measuring operator**.

2.3 The potential equation

Looking at the light propagation as an interaction between an emitter and a receiver, the radiance describes this interaction from the point of view of the emitter. The same phenomenon can also be explained from the point of view of the receiver, when the appropriate measure is called the **potential**. By definition, the potential $W(\vec{y}, \omega')$

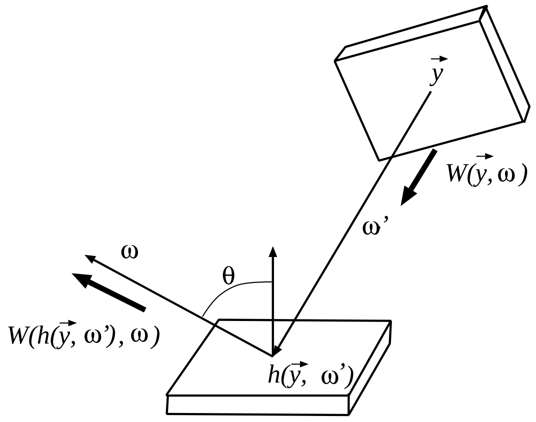


Figure 2.5: Geometry of the potential equation

expresses the effect of that portion of the unit power light-beam emitted by point $\vec{y}$ in direction ω', which actually lands at a given measuring device either directly or indirectly after some reflections or refractions. Using probabilistic terms the potential is the product of the scaling value C and the probability that a light-beam emitted by a point in a given direction reaches the measuring device.

If only **direct contribution** is considered, then $W(\vec{y}, \omega') = W^e(\vec{y}, \omega')$. To take into account light reflections, we can establish the **potential equation** [PM95]:

$$W = W^e + \mathcal{T}'W. \tag{2.19}$$

In this equation integral operator $\mathcal{T}'$ — which is the adjoint of $\mathcal{T}$ — describes the potential transport

$$(\mathcal{T}'W)(\vec{y}, \omega') = \int_{\Omega} W(h(\vec{y}, \omega'), \omega) \cdot f_r(\omega', h(\vec{y}, \omega'), \omega) \cdot \cos\theta \, d\omega, \tag{2.20}$$

where θ is the angle between the surface normal and the outgoing direction ω.

To prove this equation, we can use the probabilistic interpretation of the potential. Denoting the *"unit power light-beam that is leaving $\vec{x}$ at direction ω"* by $\Pi(\vec{x}, \omega)$, we can write:

$$W(\vec{y}, \omega') = C \cdot \Pr\{\Pi(\vec{y}, \omega') \text{ lands at the device}\} =$$

$$C \cdot \Pr\{\Pi(\vec{y}, \omega') \text{ goes directly to the device}\} + C \cdot \Pr\{\Pi(\vec{y}, \omega') \text{ lands at the device after at least one reflection}\}. \tag{2.21}$$

The first term is equal to the sensitivity function. Considering the second term, we can obtain:

$$C \cdot \Pr\{\Pi(\vec{y}, \omega') \text{ lands at the device after at least one reflection}\} =$$

$$\int_{\Omega} C \cdot \Pr\{\Pi(h(\vec{y}, \omega'), \omega) \text{ lands at the device}\} \cdot \Pr\{\Pi(\vec{y}, \omega') \text{ is reflected to } d\omega \text{ around } \omega \text{ at } h(\vec{y}, \omega')\} =$$

$$\int_{\Omega} W(h(\vec{y}, \omega'), \omega) \cdot f_r(\omega', h(\vec{y}, \omega'), \omega) \cdot \cos\theta \, d\omega. \tag{2.22}$$

2.4 Measuring the potential

Alternatively to the radiance, the power arriving at the measuring device can also be computed from the potential. Since

$$d\Phi^e(\vec{y}, \omega') = L^e(\vec{y}, \omega') \cdot \cos\theta \, d\vec{y} \, d\omega'$$

is proportional to the power of the light-beam emitted by $d\vec{y}$ in $d\omega'$ and the probability that the photons of this beam really go either directly or indirectly to the measuring device is $W(\vec{y}, \omega')/C$, the total measured value that takes into account all possible emission points and directions is

$$C \cdot \int_{\Omega} \int_{S} d\Phi^e(\vec{y}, \omega') \cdot \frac{W(\vec{y}, \omega')}{C} = \int_{\Omega} \int_{S} W(\vec{y}, \omega') \cdot L^e(\vec{y}, \omega') \cdot \cos\theta \, d\vec{y} \, d\omega' = \mathcal{M}'W, \tag{2.23}$$

where $\mathcal{M}'$ is the **potential measuring operator**. Note that unlike the radiance measuring operator, the potential measuring operator integrates on the lightsource.

2.5 The rendering problem

Generally, the **rendering problem** is a quadruple [Kel97]:

$$\langle S, f_r(\omega', \vec{x}, \omega), L^e(\vec{x}, \omega), \mathbf{W}^e(\vec{x}, \omega)\rangle$$

where S is the geometry of surfaces, f_r is the BRDF of surface points, L^e is the emitted radiance of surface points at different directions and $\mathbf{W}^e$ is a collection of measuring functions. Rendering algorithms aim at modeling and simulation of light-surface interactions to find out the power emitted by the surfaces and landing at the measuring devices. The light-surface interaction can be formulated by the *rendering equation* or alternatively by the *potential equation*. Since according to the definition of the radiance the total power of the scene is the integral of the radiance function

$$\Phi = \int\limits_{\Omega} \int\limits_{S} L(\vec{y}, \omega)\, d\vec{y}\, d\vec{\omega},$$

we are interested only in those solutions where the total integral of the radiance function is finite. Formally the solution is searched in the L_1 function space.

Recall that the measured value can be computed by the measuring function from the radiance

$$\int\limits_{\Omega} \int\limits_{S} L(\vec{y}, \omega) \cos\theta \cdot W^e(\vec{y}, \omega)\, d\vec{y}\, d\omega = \mathcal{M}L, \tag{2.24}$$

where $\mathcal{M}$ is the **radiance measuring operator**. Let us introduce the **scalar product** $\langle u, v\rangle$

$$\langle u, v\rangle = \int\limits_{\Omega} \int\limits_{S} u(\vec{y}, \omega) \cdot v(\vec{y}, \omega)\, d\vec{y}\, d\vec{\omega} = \int\limits_{\Omega} \int\limits_{S} u(\vec{y}, \omega) \cdot v(\vec{y}, \omega)\, d\vec{y} \cos\theta\, d\omega,$$

where θ is the angle between the surface normal and direction ω and thus $d\vec{y}\cos\theta$ is the infinitesimal visible area from direction ω. Using this scalar product, we can obtain an alternative form of the measuring operator:

$$\mathcal{M}L = \langle L, W^e\rangle.$$

The **potential measuring operator** can also be given in a scalar product form:

$$\mathcal{M}'W = \langle L^e, W\rangle. \tag{2.25}$$

Let us examine a single reflection of the light. The measured value taking into account a single reflection can be expressed in two ways:

$$\langle \mathcal{T} L^e, W^e\rangle = \langle L^e, \mathcal{T}' W^e\rangle. \tag{2.26}$$

Thus the radiance and potential transport operators are **adjoint operator**s [Mát81].

2.5.1 Geometry of the surfaces

A **surface** is a set of 3D points which are defined by an equation [SK99c]. Those points are said to be in this set, which satisfy the definition equation. The equation can produce the points on the surface either implicitly, when the general form is

$$F(x, y, z) = 0,$$

or explicitly, which is preferred in rendering algorithms. The general form of an explicit surface definition is:

$$\vec{r} = \vec{r}(u, v), \quad u \in [0,1],\ v \in [0,1]. \tag{2.27}$$

Points on the surface can be generated by selecting u, v parameters from the unit interval and substituting their values into the function $\vec{r}(u, v)$. For example, a sphere of radius R and of center (x_0, y_0, z_0) can be defined either by the following **explicit equation**:

$$x = x_0 + R \cdot \cos 2\pi u \cdot \sin \pi v, \quad y = y_0 + R \cdot \sin 2\pi u \cdot \sin \pi v, \quad z = z_0 + R \cdot \cos \pi v, \quad u, v \in [0,1],$$

or by an **implicit equation**:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - R^2 = 0.$$

Some rendering algorithms do not work with the original geometry but approximate the surfaces by **triangle mesh**es. This approximation — also called the **tessellation** — can be realized by selecting $n \times m$ points in parameter space $u \in [0,1], v \in [0,1]$ and adding the

$$[\vec{r}(u_i, v_j), \vec{r}(u_{i+1}, v_j), \vec{r}(u_{i+1}, v_{j+1})] \quad \text{and} \quad [\vec{r}(u_i, v_j), \vec{r}(u_{i+1}, v_{j+1}), \vec{r}(u_i, v_{j+1})]$$

triangles for all $i = 1 \ldots n-1$ and $j = 1 \ldots m-1$ indices to the resulting list (figure 2.6). For the discussion of surface definition methods using forward and reverse engineering and of transformations refer to [SK99c, Chi88, VMC97, RVW98] and to [SKe95, Kra89, Her91, Lan91], respectively.

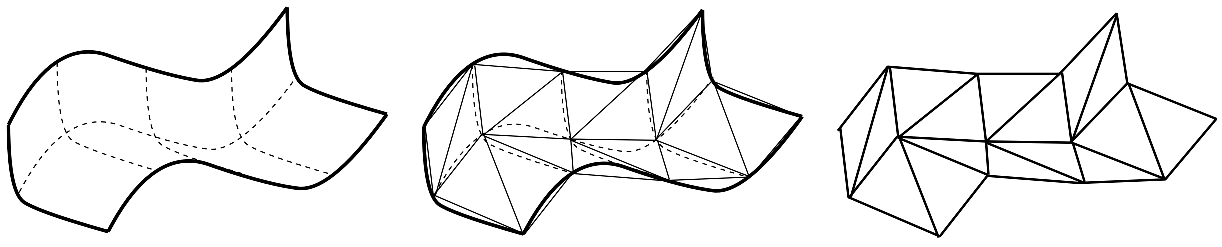

original surface with isolines
selecting points and triangles
result of tessellation

Figure 2.6: Tessellation process

2.5.2 Bi-directional Reflection Distribution Functions

The **Bi-directional Reflection Distribution Function**s (or **BRDF**s) characterize the optical material properties. Photorealistic rendering algorithms require the BRDFs not to violate physics. Such BRDF models must satisfy both reciprocity and energy balance, and are called **physically plausible** [Lew93].

Reciprocity that was recognized by Helmholtz is the symmetry property of the BRDF characterizing reflections, which is defined by the following equation [Min41]:

$$f_r(\omega, \vec{x}, \omega') = f_r(\omega', \vec{x}, \omega), \tag{2.28}$$

where ω' is the vector pointing towards the incoming light and vector ω defines the viewing direction. Reciprocity is important because it allows for the backward tracing of the light as happens in visibility ray-tracing algorithms. Note that reciprocity is valid if the incoming and outgoing beams are in the same material, that is, reflection BRDFs are expected to be reciprocal but refraction BRDFs are not.

Suppose that the surface is illuminated by a beam from direction ω'. **Energy balance** means that the number of outgoing photons cannot be more than how many were received. To examine it formally, we can introduce the **albedo** [Lew93] that is the ratio of the total reflected power and incoming power, or equivalently, the probability of the reflection to any direction. Energy balance means that the albedo cannot be greater than 1:

$$a(\vec{x}, \omega') = \Pr\{\text{photon is reflected} \mid \text{coming from } \omega'\} = \int\limits_{\Omega_H} w(\omega', \vec{x}, \omega)\, d\omega =$$

$$\int\limits_{\Omega_H} f_r(\omega', \vec{x}, \omega) \cdot \cos\theta \, d\omega \leq 1. \tag{2.29}$$

If the BRDF is reciprocal, then the albedo can also be interpreted as the optical response of the surface to homogeneous sky-light illumination of unit intensity:

$$\mathcal{T}1 = \int\limits_{\Omega_H} 1 \cdot f_r(\omega', \vec{x}, \omega) \cdot \cos\theta' \, d\omega' = a(\vec{x}, \omega). \tag{2.30}$$

Using the definition of the transfer probability density w in equation (2.8), we can obtain the following probabilistic interpretation of the BRDF:

$$f_r(\omega', \vec{x}, \omega) \cdot \cos\theta = \frac{1}{d\omega} \cdot \Pr\{\text{photon is reflected or refracted to } d\omega \text{ around } \omega \mid \text{it comes from } \omega'\}.$$

Different BRDF models are discussed in [SKe95, SK99c, SK99a, Pho75, Bli77, CT81, War92, HTSG91, ON94, Sch93, NNSK99b, NNSK99a, CSK98].

2.5.3 Lightsources

The **lightsource**s are surfaces that have non-zero L^e emission. Rendering algorithms also use abstract lightsources that are not associated with surfaces [SKe95]. The most important types of these **abstract lightsource model**s are the following:

- a **point-lightsource** is located at a given point of the virtual world and its size is zero. The direction of the illumination provided by this lightsource at a point is the vector from the point to the location of the lightsource and the intensity of the illumination decreases with the square of the distance. An example of this category is an electric bulb.

- a **directional-lightsource** is located at infinity in a given direction. The direction and the intensity of its illumination are constant everywhere. A directional-lightsource is, in fact, equivalent to an infinitely distant plane emitter. The sun can be considered as a directional-lightsource.
- **sky-light illumination** provides constant illumination at any point and at any direction.

2.5.4 Measuring devices

In order to establish models for the **camera**, the operation of the **human eye** can be analyzed when the human is looking at the real world and when he is watching the computer screen.

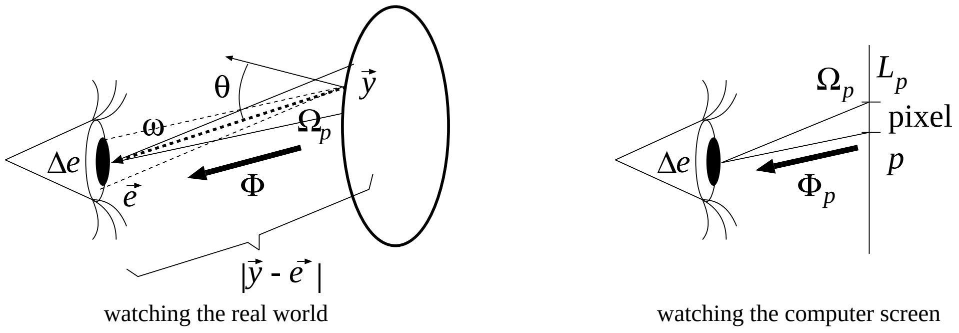


Figure 2.7: A model of the human eye

The human eye contains a lens called the **pupil** of size Δe (figure 2.7). We shall assume that the pupil is small, which is often referred as the **pinhole camera model**. When the human is watching the computer screen, a pixel p is visible in a small solid angle Ω_p. To provide the illusion that the screen is a replacement of the real world, the Φ_p power emitted by the pixel towards the eye should be similar to that Φ power which would come from the real world and would land at the pupil from solid angle Ω_p. If the emission intensity of the pixel is L_p, then the power landing at the eye from this pixel is

$$\Phi_p = L_p \cdot \Delta e \cdot \cos\theta_e \cdot \Omega_p,$$

where θ_e is the angle between the surface normal on the pupil and the direction of the pixel.

We have to find a camera model that computes a measured value P which is stored in the **image-buffer** to control the monitor. The response of **monitor**s can be characterized by a function $B \cdot \mathcal{R}$, where B represents a scaling according to the brightness settings and $\mathcal{R}$ might be non-linear. In order to compensate the monitor's non-linearity, the **look-up table** is set to distort the values of the image-buffer with the inverse of this non-linear mapping, that is by $\mathcal{R}^{-1}$. This distortion is called **gamma-correction**. Using gamma-correction, the radiant intensity of the pixel is:

$$L_p = B \cdot \mathcal{R}(\mathcal{R}^{-1}(P)) = B \cdot P.$$

Since we require that $\Phi_p = \Phi$, our model camera is expected to provide the following measured value:

$$P = \mathcal{R}\left(\mathcal{R}^{-1}\left(\frac{L_p}{B}\right)\right) = \frac{L_p}{B} = \frac{\Phi}{\Delta e \cdot \cos\theta_e \cdot \Omega_p \cdot B}. \tag{2.31}$$

Let us assign a measuring device for this pixel. This device is sensitive to those surface points that are visible in this pixel — i.e. in Ω_p — and to those directions that point from the surface points to the pupil. Mathematically, this can be represented by the following measuring function

$$W^e(\vec{y},\omega) = \begin{cases} C & \text{if } \vec{y} \text{ is visible in } \Omega_p \text{ and } \omega \text{ points from } \vec{y} \text{ through } \Delta e, \\ \\ 0 & \text{otherwise,} \end{cases} \tag{2.32}$$

where

$$C = \frac{1}{\Delta e \cdot \cos\theta_e \cdot \Omega_p \cdot B}.$$

The measuring device provides the following value:

$$P = \mathcal{M}L = \int_{\Omega}\int_{S} L(\vec{y},\omega) \cdot W^e(\vec{y},\omega) \cdot \cos\theta \; d\vec{y} d\omega. \tag{2.33}$$

Applying equation (2.2), we can use the following substitutions:

$$d\omega = d\vec{e} \cdot \frac{\cos\theta_e}{|\vec{y}-\vec{e}|^2}, \qquad d\vec{y} = \frac{|\vec{y}-\vec{e}|^2}{\cos\theta} \cdot d\omega_p,$$

where $d\vec{e}$ is a differential area on the pupil and $|\vec{y}-\vec{e}|$ is the distance between the pupil and the visible surface. Substituting these and taking advantage that the pupil Δe is small, we obtain

$$P = \int_{\Omega_p}\int_{\Delta e} L(h(\vec{e},-\omega_p),\omega_p) \cdot C \cdot \frac{\cos\theta_e}{|\vec{y}-\vec{e}|^2} \cdot \frac{|\vec{y}-\vec{e}|^2}{\cos\theta} \cdot \cos\theta \, d\vec{e} d\omega_p = \int_{\Omega_p}\int_{\Delta e} L(h(\vec{e},-\omega_p),\omega_p) \cdot C \cdot \cos\theta_e \, d\vec{e} d\omega_p \approx$$

$$\int_{\Omega_p} L(h(\vec{\text{eye}},-\omega_p),\omega_p) \cdot C \cdot \cos\theta_e \cdot \Delta e \, d\omega_p = \int_{\Omega_p} L(h(\vec{\text{eye}},-\omega_p),\omega_p) \cdot \frac{1}{\Omega_p \cdot B} \, d\omega_p, \tag{2.34}$$

where $\vec{\text{eye}}$ is the position of the very small pupil, and ω_p is the direction which points from $\vec{y}$ to the eye position.

Note that the measured value is independent of both the distance and the orientation of the visible surface! This is explained by the fact that as we move farther from a surface, although the power coming from a unit surface area decreases with the square of the distance, the area that can be visible in a given solid angle increases with the same speed. The two factors compensate each other. Similarly, when we look at a surface from a non perpendicular direction, the radiance is weighted by the cosine of this angle in the power, but the surface area that can be seen is inversely proportional to the cosine of the same angle.

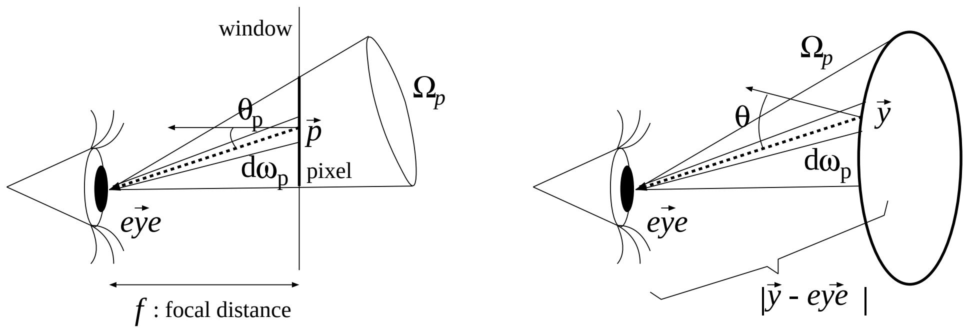


Figure 2.8: Evaluation of the measured value as an integral on the pixel (left) and on the surface (right)

Since Ω_p is the collection of those directions that go through the pixel, the measured value can also be expressed as an integral over the pixel area S_p (figure 2.8). Let $\vec{p}$ be the running point on the pixel, θ_p be the angle between the pixel normal and vector $\vec{p} - \vec{\text{eye}}$, and θ_{pix} be equal to θ_p in the center of the pixel. Since $|\vec{p} - \vec{\text{eye}}| = f/\cos\theta_p$ where f is the **focal distance**, i.e. the distance between the eye and the plane of the window, we can establish the following correspondance:

$$d\omega_p = \frac{d\vec{p} \cdot \cos\theta_p}{|\vec{p} - \vec{\text{eye}}|^2} = \frac{d\vec{p} \cdot \cos^3\theta_p}{f^2}. \tag{2.35}$$

Substituting this to equation (2.34), the measured value becomes

$$P = \int_{S_p} L(h(\vec{p},-\omega_{\vec{p}}),\omega_{\vec{p}}) \cdot \frac{1}{\Omega_p \cdot B} \cdot \frac{\cos^3\theta_p}{f^2} \, d\vec{p}. \tag{2.36}$$

Let us introduce **camera parameter** $c(\vec{p})$ by

$$c(\vec{p}) = \frac{S_p}{\Omega_p \cdot B} \cdot \frac{\cos^3\theta_p}{f^2}. \tag{2.37}$$

Since

$$\Omega_p = \int_{\Omega_p} d\omega_p = \int_{S_p} \frac{\cos^3\theta_p}{f^2} \, d\vec{p} \approx \frac{S_p \cdot \cos^3\theta_{\text{pix}}}{f^2}, \tag{2.38}$$

the camera parameter can also be expressed in the following exact and approximating forms:

$$c(\vec{p}) = \frac{S_p \cdot \cos^3\theta_p}{B \cdot \int_{S_p} \cos^3\theta_p \, d\vec{p}} \approx \frac{\cos^3\theta_p}{B \cdot \cos\theta^3_{\text{pix}}} \approx \frac{1}{B}. \tag{2.39}$$

The approximations are accurate if the pixel is small compared to the focal distance. If image space **filtering** is also applied, then S_p may be greater than the pixel and the camera parameter is not necessarily constant but follows the shape of a reconstruction filter [SKe95, SK95]. Summarizing the measured value is an integral on the pixel:

$$P = \int_{S_p} L(h(\vec{p}, -\omega_{\vec{p}}), \omega_{\vec{p}}) \cdot \frac{c(\vec{p})}{S_p} \, d\vec{p}. \tag{2.40}$$

Equation (2.34) can also be evaluated on object surfaces. Using the notations of figure 2.8, we can obtain:

$$d\omega_p = \frac{d\vec{y} \cdot \cos\theta}{|\vec{y} - \vec{\text{eye}}|^2} \tag{2.41}$$

where $\vec{y}$ is a surface point visible in the pixel. Let the indicator function of those points that are visible in Ω_p be $V_{\Omega_p}(\vec{y})$. The measured value is then:

$$P = \int_{S} L(\vec{y}, \omega_{\vec{y}\to\vec{\text{eye}}}) \cdot V_{\Omega_p}(\vec{y}) \cdot \frac{1}{\Omega_p \cdot B} \cdot \frac{\cos\theta}{|\vec{y} - \vec{\text{eye}}|^2} \, d\vec{y}. \tag{2.42}$$

Let us introduce **surface dependent camera parameter** $g(\vec{y})$ by

$$g(\vec{y}) = \frac{1}{\Omega_p \cdot B \cdot |\vec{y} - \vec{\text{eye}}|^2} = \frac{f^2}{B \cdot |\vec{y} - \vec{\text{eye}}|^2 \cdot \int_{S_p} \cos^3\theta_p \, d\vec{p}} \approx \frac{f^2}{B \cdot |\vec{y} - \vec{\text{eye}}|^2 \cdot S_p \cdot \cos^3\theta_{\text{pix}}} \tag{2.43}$$

using formula (2.38) that expresses Ω_p. Thus the final form of the measured value is:

$$P = \int_{S} L(\vec{y}, \omega_{\vec{y}\to\vec{\text{eye}}}) \cdot V_{\Omega_p}(\vec{y}) \cdot g(\vec{y}) \cdot \cos\theta \, d\vec{y}. \tag{2.44}$$

Comparing this formula with equation (2.33), the following sensitivity function can be derived:

$$W^e(\vec{y}, \omega) = \begin{cases} \delta(\omega - \omega_{\vec{y}\to\vec{\text{eye}}}) \cdot g(\vec{y}) & \text{if } \vec{y} \text{ is visible through the pixel,} \\ 0 \quad \text{otherwise.} \end{cases} \tag{2.45}$$

2.6 Numerical solution of the rendering equation

2.6.1 Error measures for numeric techniques

When solving the rendering problem numerically, the required radiance, potential or measured power can only be approximated. To characterize the accuracy of the approximation, **error measure**s are needed. The errors of radiance or potential are called **object-space error measure**s. The error of the measured power is called **image-space error measure**. A good error measure should be invariant to changes that modify neither the object space nor the camera. Particularly, the error should not change when a surface is subdivided to two surfaces or the complete scene is scaled — i.e. it should be tessellation and scaling independent —, or when a pixel is subdivided to increase the image resolution — i.e. it should be resolution independent.

Tessellation and scaling independent measures can be defined if the norm of the error function is divided by the total surface area. The **norm** [Mát81, ST93] is selected from the following possibilities:

$$||f||_1 = \int_{\Omega}\int_{S} |f(\vec{x}, \omega)| \, d\vec{x} \, d\vec{\omega}, \quad ||f||_2 = \sqrt{\int_{\Omega}\int_{S} (f(\vec{x}, \omega))^2 \, d\vec{x} \, d\vec{\omega}}, \quad ||f||_\infty = \max_{\vec{x}, \omega} |f(\vec{x}, \omega)|. \tag{2.46}$$

Using any of these, if L is the calculated radiance and $\tilde{L}$ is the real radiance, then an absolute object-space error ϵ_a and a relative object-space error ϵ_r are

$$\epsilon_a = \frac{||L - \tilde{L}||}{S}, \quad \epsilon_r = \frac{||L - \tilde{L}||}{||\tilde{L}||}. \tag{2.47}$$

Image space norms are based on the powers that go through different pixels $p = 1, 2, \ldots, N_P$.

$$||\Phi||_1 = \sum_{p=1}^{N_P} |\Phi_p|, \quad ||\Phi||_2 = \sqrt{\sum_{p=1}^{N_P} |\Phi_p|^2}, \quad ||\Phi||_\infty = \max_p |\Phi_p|. \tag{2.48}$$

Resolution independent measures divide the total error of pixels by the number of pixels (P). If the calculated and real powers that go through pixel p are Φ_p and $\tilde{\Phi}_p$, respectively, then an absolute and a relative image space error measures are

$$\epsilon_a = \frac{||\Phi - \tilde{\Phi}||}{N_P}, \quad \epsilon_r = \frac{||\Phi - \tilde{\Phi}||}{||\tilde{\Phi}||}. \tag{2.49}$$

2.6.2 Properties of the rendering equation

The integral operators of both the rendering and the potential equations are **contraction**s. This statement can be proven from the fact that for physically plausible models, the albedo is less than 1. Here, the ∞-norm is used:

$$\|\mathcal{T}L\|_\infty = \max|\mathcal{T}L| \leq \max|L| \cdot \max|\mathcal{T}1| = \max|L| \cdot \max|a(\vec{x}, \omega)| = \|L\|_\infty \cdot \max|a(\vec{x}, \omega)|.$$

For the potential, similar results can be obtained:

$$\|\mathcal{T}'W\|_\infty = \max|\mathcal{T}'W| \leq \max|W| \cdot \max|\mathcal{T}'1| = \max|W| \cdot \max|a(h(\vec{y}, \omega'), \omega')| =$$

$$\|W\|_\infty \cdot \max|a(\vec{x}, \omega)|.$$

2.7 Classification of the solution techniques

In order to find the color of a pixel, the radiance has to be evaluated for that surface which is visible through the given pixel. The determination of this surface is called the **hidden surface problem** or **visibility calculation**. Having solved the visibility problem, the surface visible in the given pixel is known, and the radiance may be calculated on the representative wavelengths by the rendering equation.

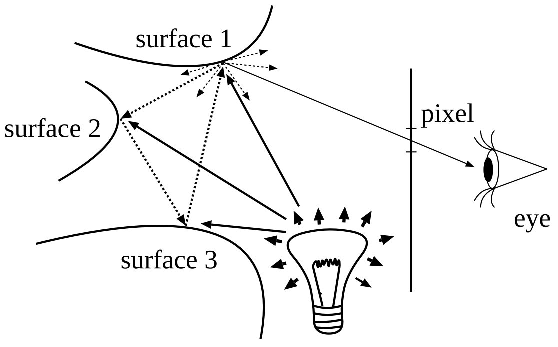


Figure 2.9: Multiple reflections of light

Due to multiple reflections of light beams, the calculation of the radiance of the light leaving a point on a surface in a given direction requires the radiances of other surfaces visible from this point, which generates new visibility and shading problems to solve (figure 2.9). To calculate those radiances, other surfaces should be evaluated, and our original point on the given surface might contribute to those radiances. As a consequence of that, the rendering equation has complicated coupling between its left and right sides, making the evaluation difficult.

There are three general and widely accepted approaches to attack this coupling:

1. **Local illumination methods**

   Local illumination methods take a drastic approach and simplify or disregard completely all the terms causing problems. The unknown radiance inside the integral of the rendering equation is replaced by some approximation of the emission function. Formally, these methods evaluate the following simplified rendering equation to obtain the radiance:

$$L(\vec{x}, \omega) = L^e(\vec{x}, \omega) + \int_\Omega L_{\text{lightsource}}(h(\vec{x}, -\omega'), \omega') \cdot f_r(\omega', \vec{x}, \omega) \cdot \cos\theta' \, d\omega', \tag{2.50}$$

where $L_{\rm lightsource}$ may be a simplification of the emission function L^e. Abstract lightsources, such as point or directional lightsources are preferred here, since their radiance is a Dirac-delta like function, which simplifies the integral of equation (2.50) to a sum.

These methods take into account only a single reflection of the light coming from the abstract lightsources. Ideal mirrors and refracting objects cannot be rendered with these methods.

2. **Recursive ray-tracing**

   Another alternative is to eliminate from the rendering equation those energy contributions which cause the difficulties, and thus give ourselves a simpler problem to solve. For example, if limited level, say n, coupling caused by ideal reflection and refraction were allowed, and we were to ignore the other non-ideal components coming from non-abstract lightsources, then the number of surface points which would need to be evaluated to calculate a pixel color can be kept under control. Since the illumination formula contains two terms regarding the coherent components (reflective and refracting lights), the maximum number of surfaces involved in the color calculation of a pixel is two to the power of the given depth, i.e. 2^n. An implementation of this approach is called **visibility ray-tracing** which uses the following illumination formula:

$$L(\vec{x},\omega) = L^e(\vec{x},\omega) + \int\limits_{\Omega} L_{\rm lightsource}(h(\vec{x},-\omega'),\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta'\, d\omega' +$$

$$k_r(\omega_r,\vec{x},\omega) \cdot L(h(\vec{x},-\omega_r),\omega_r) + k_t(\omega_t,\vec{x},\omega) \cdot L(h(\vec{x},-\omega_t),\omega_t), \qquad (2.51)$$

   where ω_r and ω_t are the ideal reflection and refraction directions, and k_r and k_t are the integrated BRDF components representing the ideal reflection and refraction, respectively.

3. **Global illumination solution**

   Local illumination and recursive ray-tracing apply brutal simplifications and thus provide physically inaccurate images. Thus these methods cannot be reliably used in engineering applications, such as in lighting design. These application areas require the rendering equation to be solved without relevant simplifications, that leads to the family of **global illumination** methods. Global illumination methods rely on numerical techniques to resolve the coupling in the rendering or potential equations and to solve the integral equation without unacceptable simplifications.

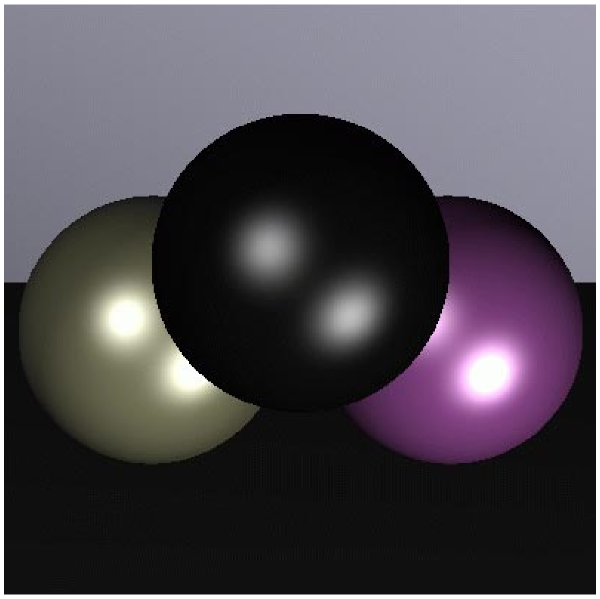
The three different approaches represent three levels of the compromise between image generation speed and quality. By ignoring more and more terms in the illumination formula, its calculation can be speeded up, but the result inevitably becomes more and more artificial.

The ignoration of the terms, however, violates the energy equilibrium, and causes portions of objects to come out extremely dark, sometimes unexpectedly so. These artifacts can be reduced by reintroducing the ignored terms in simplified form, called **ambient light**. The ambient light represents the ignored energy contribution in such a way as to satisfy the energy equilibrium. Since this ignored part is not calculated, nothing can be said of its positional and directional variation, hence it is supposed to be constant in all directions and everywhere in the 3D space. From this aspect, the role of ambient light also shows the quality of the shading algorithm. The more important a role it has, the poorer quality picture it will generate.

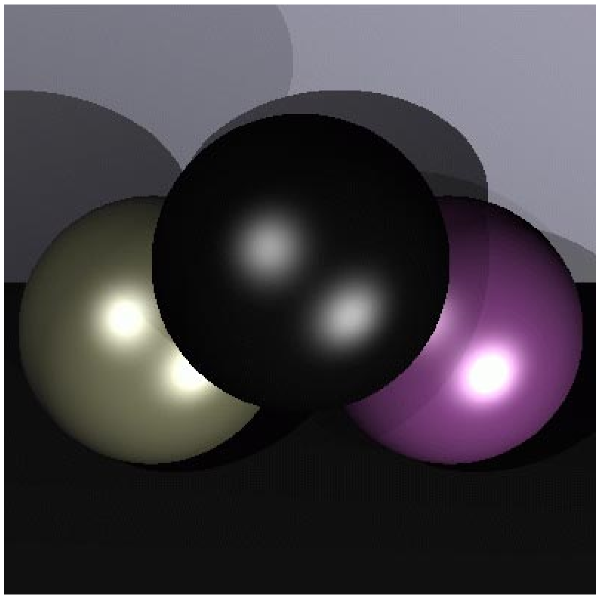
In order to formally express the capabilities of a certain algorithm, Heckbert has developed a notation that is based on the regular expressions originally proposed for the syntax of formal languages [Hec91]. The elements of the notation are as follows:

- E is the eye,
- L is the lightsource,
- D is a non-ideal — i.e. incoherent — reflection or refraction,
- S is an ideal reflection or refraction,
- * is the sign of iteration, that is, it can mean $0, 1, 2, \ldots$ applications of the given operator,
- [ ] represents optionality,
- | means selection.

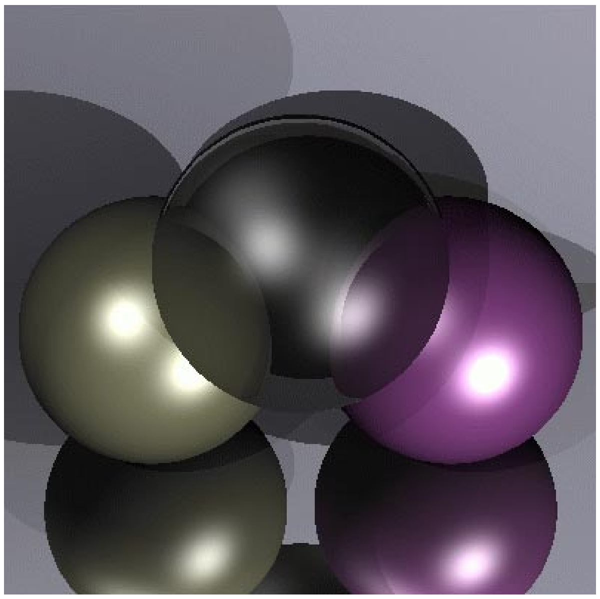
A global illumination algorithm is expected to model all types of lightpaths, that is, it must have $L[D|S]^*E$ type. Visibility ray-tracing allows multiple steps only for the ideal reflection or refraction, thus it can model only $L[DS^*]E$ path. Finally, local illumination models simulate only a single non-ideal reflection and fall into the $L[D]E$ category.



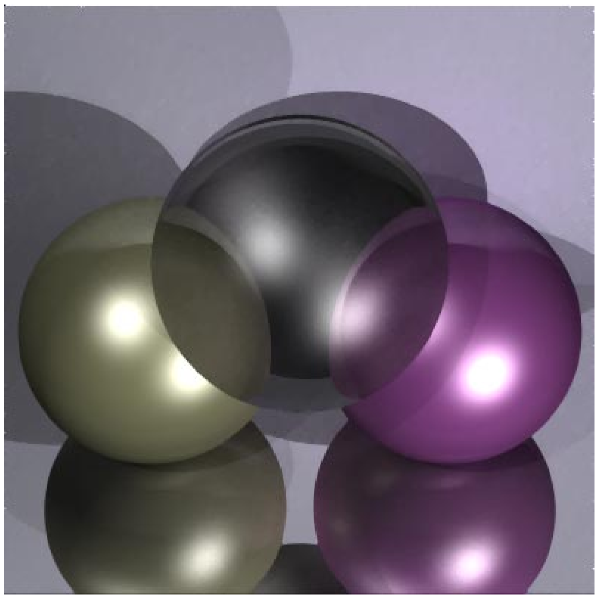
local illumination method



local illumination method with shadow computation



recursive ray-tracing



global illumination method

Figure 2.10: Comparison of local illumination method, recursive ray-tracing and global illumination method. Ambient light was also added when the local illumination method and recursive ray-tracing were computed. The images have been generated by a path tracing program [SK99c]. Note that local illumination methods cannot render ideal reflection or refraction, thus there are no transparent and mirror like objects. Recursive ray-tracing, on the other hand, is unable to follow multiple non-ideal reflections, thus the illumination diffusely reflected from the back wall is missing on the spheres. The running times are 90 seconds, 95 seconds, 135 seconds, 9 hours, respectively, which clearly shows the price we should pay for a physically accurate simulation.

Chapter 3

Optical material models

In order to render realistic images, we have to use realistic material models, also called BRDFs. A realistic BRDF model is expected not to violate physics laws including the **Helmholtz-symmetry**, and the law of **energy conservation**, and to mimic the features of real materials.

The Helmholtz principle states that the incoming and outgoing directions can be exchanged in the BRDF:

$$f_r(\omega, \vec{x}, \omega') = f_r(\omega', \vec{x}, \omega). \tag{3.1}$$

According to energy conservation, a surface — provided that it is not a lightsource — cannot reflect more photons (more power) than what have been received, thus the albedo defined by

$$a(\vec{x}, \omega') = \int\limits_{\Omega_H} f_r(\omega', \vec{x}, \omega) \cdot \cos\theta \, d\omega$$

cannot be greater than 1. For **isotropic model**s, reflections of illuminations from incident directions having the same θ' have rotational symmetry, thus the albedo of a given point depends only on incident angle θ'. To emphasize this, the albedo will be denoted by $a(\theta')$.

When introducing the BRDF models, the following notations are used: $\vec{N}$ is the unit normal vector of the surface, $\vec{L}$ is a unit vector pointing towards the lightsource, $\vec{V}$ is a unit vector pointing towards the camera, $\vec{R}$ is the mirror vector of $\vec{L}$ onto $\vec{N}$, $\vec{H}$ is a unit vector that is halfway between $\vec{L}$ and $\vec{V}$, θ' is the angle between $\vec{L}$ and $\vec{N}$, θ is the angle between $\vec{V}$ and $\vec{N}$, and ψ is the angle between $\vec{R}$ and $\vec{V}$.

3.1 Diffuse reflection

First of all, consider diffuse — optically very rough — surfaces reflecting a portion of the incoming light with radiance uniformly distributed in all directions. Looking at the wall, sand, etc. the perception is the same regardless of the viewing direction.

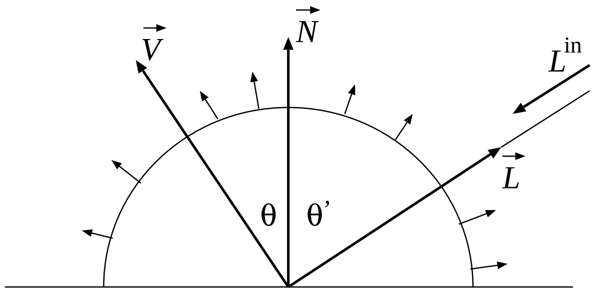


Figure 3.1: Diffuse reflection

If the BRDF is independent of the viewing direction, it must also be independent of the light direction because of the Helmholtz-symmetry, thus the BRDF of these **diffuse** surfaces is constant:

$$f_{r,\text{diffuse}}(\vec{L}, \vec{V}) = k_d. \tag{3.2}$$

The albedo of diffuse reflection is

$$a_{\text{diffuse}}(\theta') = \int_{\Omega_H} k_d \cdot \cos\theta \, d\omega = \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\pi/2} k_d \cdot \cos\theta \cdot \sin\theta \, d\theta d\phi = k_d \cdot \pi. \tag{3.3}$$

Thus for energy conservation

$$k_d \leq 1/\pi$$

should hold.

3.2 Ideal, mirror-like reflection

An **ideal mirror** complies the **reflection law** of **geometric optics**, which states that incoming direction $\vec{L}$, surface normal $\vec{N}$ and outgoing direction $\vec{R}$ are in the same plane, and incoming angle θ' equals to outgoing angle θ ($\theta' = \theta$). An ideal mirror reflects only into the ideal reflection direction $\vec{R}$. The BRDF can thus be defined by a **Dirac-delta** function:

$$f_{r,\text{mirror}}(\vec{L}, \vec{V}) = k_r(\theta') \cdot \frac{\delta(\vec{R} - \vec{V})}{\cos\theta'}. \tag{3.4}$$

The albedo of ideal reflection is

$$a_{\text{mirror}}(\theta') = \int_{\Omega_H} k_r(\theta') \cdot \frac{\delta(\vec{R} - \vec{V})}{\cos\theta'} \cdot \cos\theta \, d\omega = k_r(\theta'). \tag{3.5}$$

Thus for energy conservation k_r cannot exceed 1.

Even perfect mirrors absorb some portion of the incident light, as is described by the **Fresnel equations** of physical optics, expressing the reflection (F) of a perfectly smooth mirror in terms of the refractive index of the material n, the extinction coefficient κ which describes the conductivity of the material (for nonmetals $\kappa = 0$), and the angle of the incidence of the light beam. Denoting the incident angle by θ' and the angle of refraction by θ, the

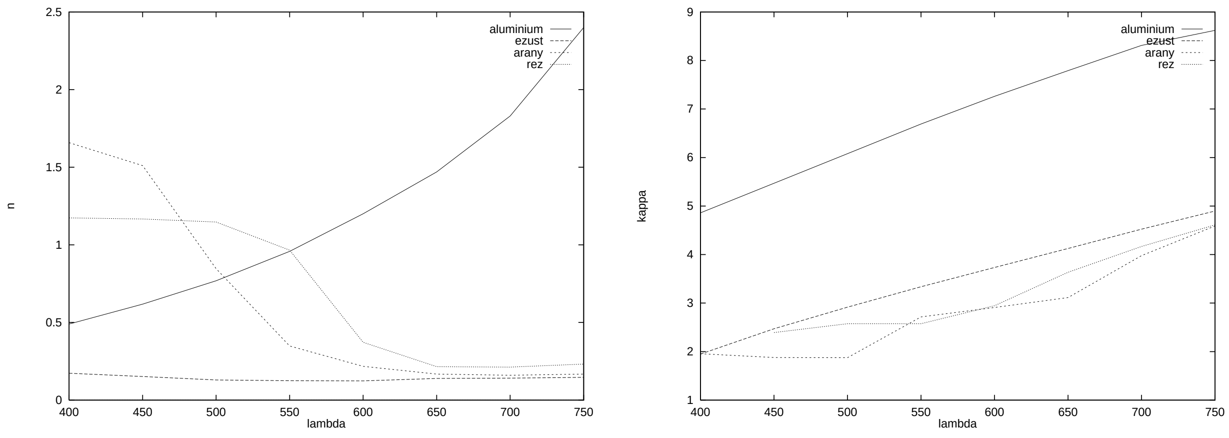


Figure 3.2: Refraction index of the gold, copper and silver

Fresnel equation expressing the ratio of the energy of the reflected beam and the energy of the incident beam for the directions parallel and perpendicular to the electric field is:

$$F_\perp(\lambda, \theta') = \left| \frac{\cos\theta - (n + \kappa\jmath) \cdot \cos\theta'}{\cos\theta + (n + \kappa\jmath) \cdot \cos\theta'} \right|^2, \quad F_\parallel(\lambda, \theta') = \left| \frac{\cos\theta' - (n + \kappa\jmath) \cdot \cos\theta}{\cos\theta' + (n + \kappa\jmath) \cdot \cos\theta} \right|^2, \tag{3.6}$$

where $\jmath = \sqrt{-1}$. These equations can be derived from Maxwell's fundamental formulae describing the basic laws of electric waves. If the light is unpolarized, that is, the parallel ($\vec{E}_\parallel$) and the perpendicular ($\vec{E}_\perp$) electric fields have the same amplitude, then the total reflectivity is:

$$k_r(\theta') = F(\lambda, \theta') = \frac{|F_\parallel^{1/2} \cdot \vec{E}_\parallel + F_\perp^{1/2} \cdot \vec{E}_\perp|^2}{|\vec{E}_\parallel + \vec{E}_\perp|^2} = \frac{F_\parallel + F_\perp}{2}. \tag{3.7}$$

Note that F is wavelength dependent, since n and κ are functions of the wavelength. Parameters n and κ are often not available, so they should be estimated from measurements, or from the value of the normal reflection if the extinction is small. At normal incidence ($\theta' = 0$), the reflection is:

$$F_0(\lambda) = \left|\frac{1-(n+\kappa\jmath)}{1+(n+\kappa\jmath)}\right|^2 = \frac{(n-1)^2+\kappa^2}{(n+1)^2+\kappa^2} \approx \left[\frac{n-1}{n+1}\right]^2. \tag{3.8}$$

Solving for n gives the following equation:

$$n(\lambda) \approx \frac{1+\sqrt{F_0(\lambda)}}{1-\sqrt{F_0(\lambda)}}. \tag{3.9}$$

F_0 can easily be measured, thus this simple formula is used to compute the values of the index of refraction n. Values of $n(\lambda)$ can then be substituted into the Fresnel equations to obtain reflection parameter F for other angles of incidence.

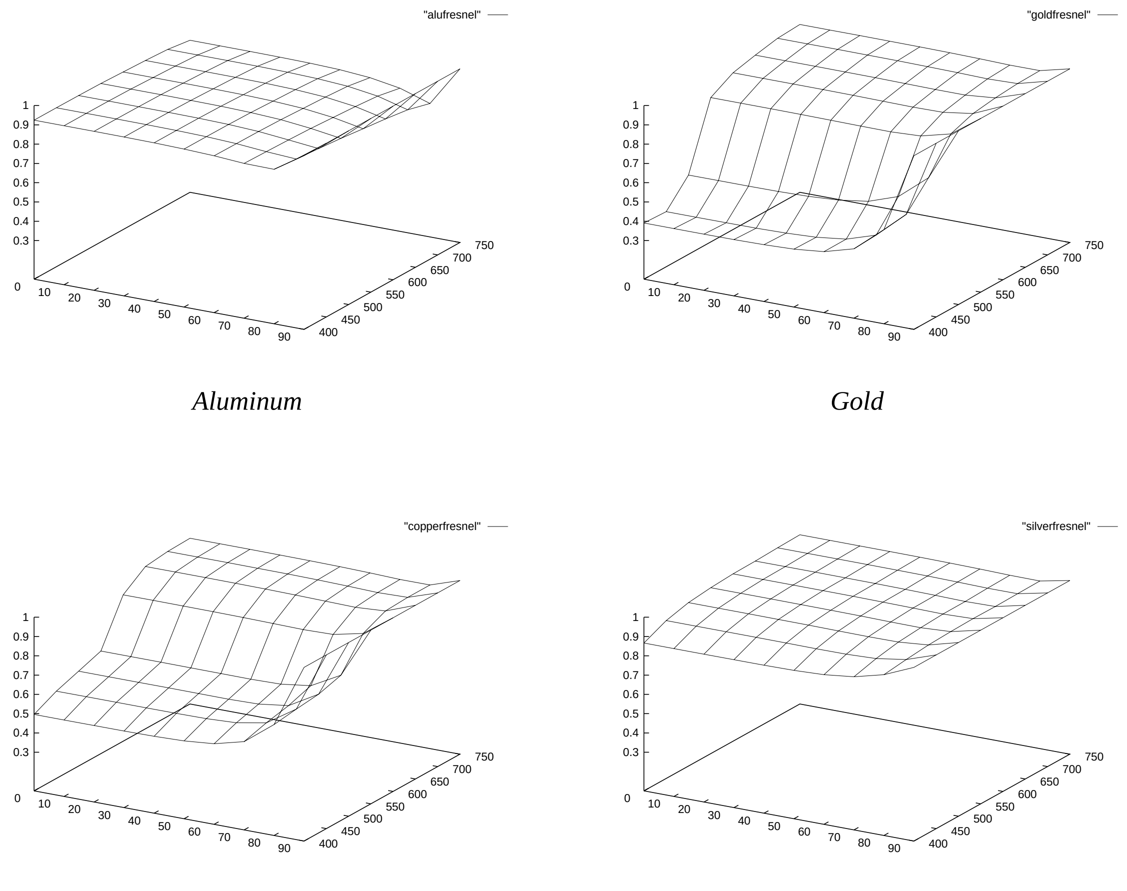


Figure 3.3: The Fresnel function of aluminum, gold, copper and silver for different wavelengths (400..700 nm) and incident angles (0..90 degrees)

3.3 Ideal refraction

When ideal refraction happens, the light goes into the material according to the **Snellius-Descartes-law**, which states that the incident direction ($\vec{L}$), surface normal ($\vec{N}$) and the refraction direction ($\vec{V}$) are all in the same plane, and

$$\frac{\sin\theta'}{\sin\theta} = n,$$

where n is the relative refraction index of the material. Thus the BRDF of ideal refraction is also a Dirac-delta like function:

$$f_{r,\text{refractor}}(\vec{L},\vec{V}) = k_t \cdot \frac{\delta(\vec{T}-\vec{V})}{\cos\theta'}, \tag{3.10}$$

where $\vec{T}$ is the ideal refraction direction.

3.4 Non-ideal, specular reflection

Incoherent — i.e. not ideal, mirror-like — reflection is usually broken down into diffuse and specular components. The BRDF of the diffuse reflection has already been discussed. For the **specular reflection**, new models are proposed.

3.4.1 Phong reflection model and its modifications

The **Phong BRDF** [Pho75] was the first model proposed for specular materials. Specular surfaces reflect most of the incoming light around the ideal reflection direction, thus the BRDF should be maximum at this direction and should decrease sharply.

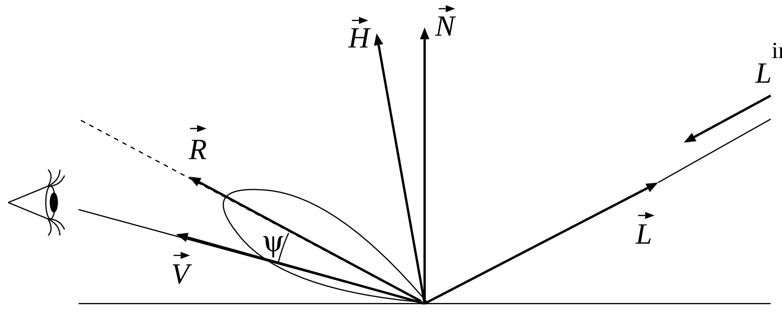


Figure 3.4: Specular reflection

Phong proposed the following function for this purpose:

$$f_{r,\text{Phong}}(\vec{L},\vec{V}) = k_s \cdot \frac{\cos^n\psi}{\cos\theta'} = k_s \cdot \frac{(\vec{R}\cdot\vec{V})^n}{(\vec{N}\cdot\vec{L})} \tag{3.11}$$

where $\vec{R}$ is the mirror direction of $\vec{L}$ onto the surface normal.

The k_s factor is proportional to the Fresnel function, but is less than that, because now the surface is not an ideal mirror. Factor k_s can be assumed to be independent of the wavelength and the incident angle for non-conducting materials (the highlight of a white lightsource on plastics is also white).

The original Phong model is not physically plausible since it is not symmetric. Due to this, in photorealistic image synthesis the following **reciprocal Phong BRDF** is preferred [ICG86]:

$$f_{r,\text{reciprocalPhong}}(\vec{L},\vec{V}) = k_s \cdot \cos^n\psi = k_s \cdot (\vec{R}\cdot\vec{V})^n \tag{3.12}$$

To compute the albedo of the reciprocal Phong model, an appropriate parameterization of the directional sphere is needed (figure 3.5). Now the north pole of the spherical coordinate system is the ideal reflection direction $\vec{R}$. Let us identify a direction by an angle from $\vec{R}$, and by another angle ϕ between its projection onto a plane perpendicular to $\vec{R}$ and an arbitrary vector on this plane. When the viewing direction is parameterized in this way, the vertical angle is just ψ.

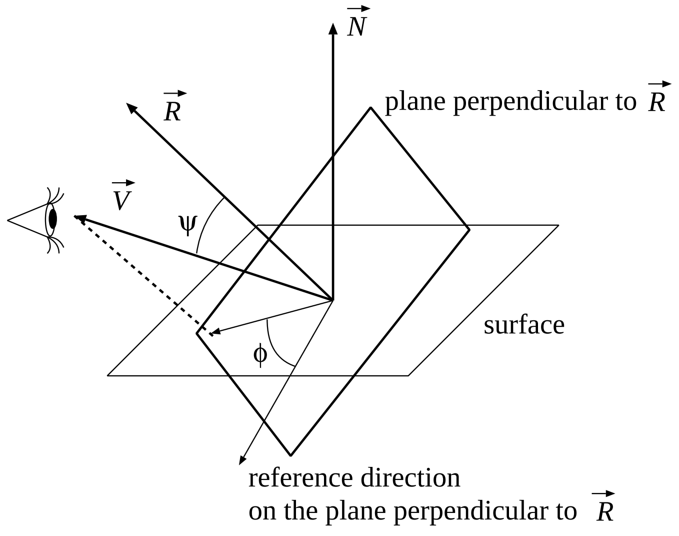

$\vec{N}$
$\vec{R}$
plane perpendicular to $\vec{R}$
$\vec{V}$
ψ
surface
ϕ
reference direction
on the plane perpendicular to $\vec{R}$

Figure 3.5: Parameterization for the calculation of the albedo

Unfortunately, the domain of (ψ, ϕ) is rather complicated since those pairs that would identify a direction pointing into the object should be excluded from reflection directions. Without this exclusion the albedo is overestimated. Note that this overestimation can only double the albedo at the worst case which happens at grazing angles when half of the reflection lobe is cut away by the surface. Representing this effect by a function $\mathrm{Cut}(\theta')$, the integral computing the albedo is:

$$a_{\mathrm{reciprocalPhong}}(\theta') = \mathrm{Cut}(\theta') \cdot \int_{\phi=0}^{2\pi} \int_{\psi=0}^{\pi/2} k_s \cdot \cos^n \psi \cdot \cos\theta \cdot \sin\psi \; d\psi d\phi.$$

Function $\mathrm{Cut}(\theta')$ is 1 at perpendicular illumination and decreases to a half at horizontal illumination. When the illumination is perpendicular, then $\psi = \theta$, thus the albedo for this case is:

$$a_{\mathrm{reciprocalPhong}}(0) = \int_{\phi=0}^{2\pi} \int_{\psi=0}^{\pi/2} k_s \cdot \cos^n \psi \cdot \cos\psi \cdot \sin\psi \; d\psi d\phi = 2\pi k_s \cdot \int_{\psi=0}^{\pi/2} \cos^{n+1} \psi \cdot \sin\psi \; d\psi = k_s \cdot \frac{2\pi}{n+2}.$$

Since the albedo is less than this for other illumination angles, energy conservation requires the following relation to hold [LW94]:

$$k_s \leq \frac{n+2}{2\pi}. \tag{3.13}$$

Considering the estimation of the albedo for other illumination angles, we have to realize that even if function $\mathrm{Cut}(\theta')$ is approximated by its maximum, the $\cos\theta$ factor still forbids the symbolic integration. If the surface is very shiny — n is large — then the reflection lobe is narrow, thus $\cos\theta$ is approximately constant and is equal to $\cos\theta'$ where $\cos^n\psi$ is not negligible, thus we can further obtain:

$$a_{\mathrm{reciprocalPhong}}(\theta') \approx \cos\theta' \cdot \int_{\phi=0}^{2\pi} \int_{\psi=0}^{\pi/2} k_s \cdot \cos^n \psi \cdot \sin\psi \; d\psi d\phi = k_s \cdot \frac{2\pi}{n+1} \cdot \cos\theta'.$$

The reflected radiance of this model goes to zero for larger incident angles, which is not realistic when rendering metals (left of figure 3.6). This artifact is eliminated in the following **max-Phong BRDF** [NNSK98b]:

$$f_{r,\mathrm{maxPhong}}(\vec{L}, \vec{V}) = k_s \cdot \frac{\cos^n \psi}{\max(\cos\theta, \cos\theta')} = k_s \cdot \frac{(\vec{R} \cdot \vec{V})^n}{\max((\vec{N} \cdot \vec{V}), (\vec{N} \cdot \vec{L}))} \tag{3.14}$$

The albedo of this max-Phong model is maximum at perpendicular illumination when $\theta' = 0$ and $\theta = \psi$, thus we obtain

$$a_{\mathrm{maxPhong}}(0) = \int_{\phi=0}^{2\pi} \int_{\psi=0}^{\pi/2} k_s \cdot \cos^n \psi \cdot \sin\psi \cdot \frac{\cos\psi}{\max(\cos\psi, 1)} \, d\psi d\phi = k_s \cdot \frac{2\pi}{n+2}.$$

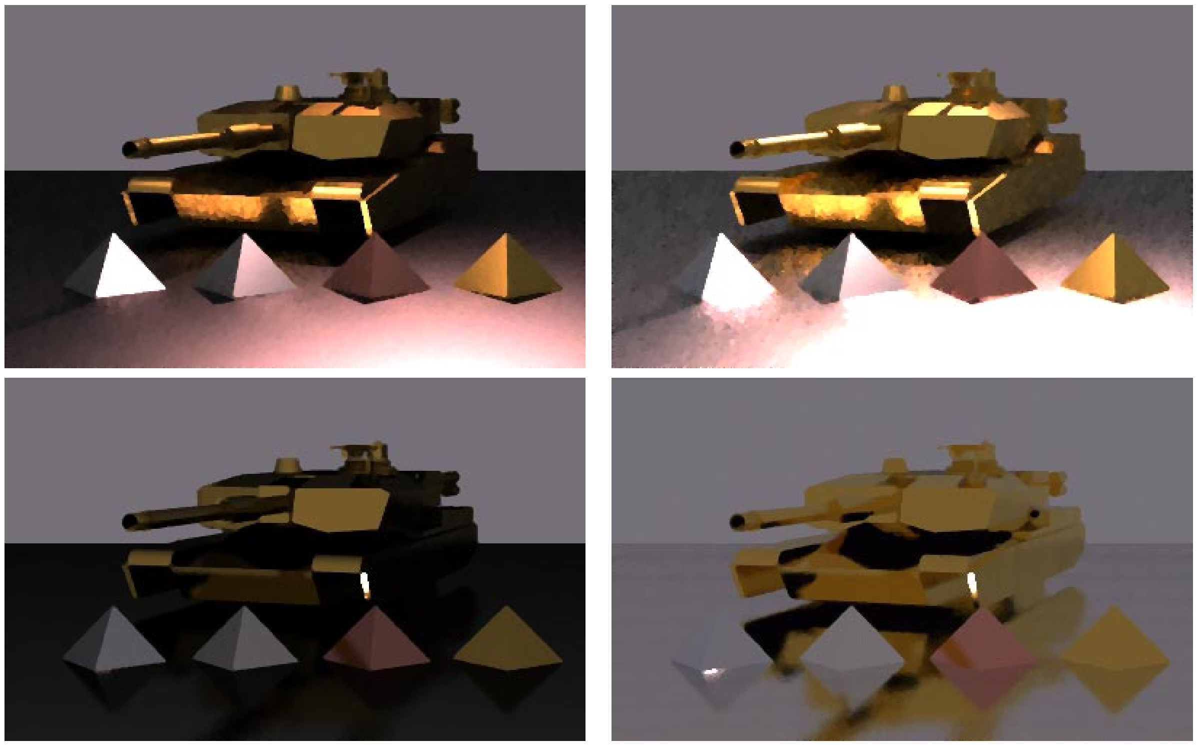


Figure 3.6: Comparison of the reciprocal Phong (left) and the max-Phong (right) models ($n = 20$ in the upper row, and $n = 5000$ in the lower row). The reciprocal Phong model becomes dark for grazing angles.

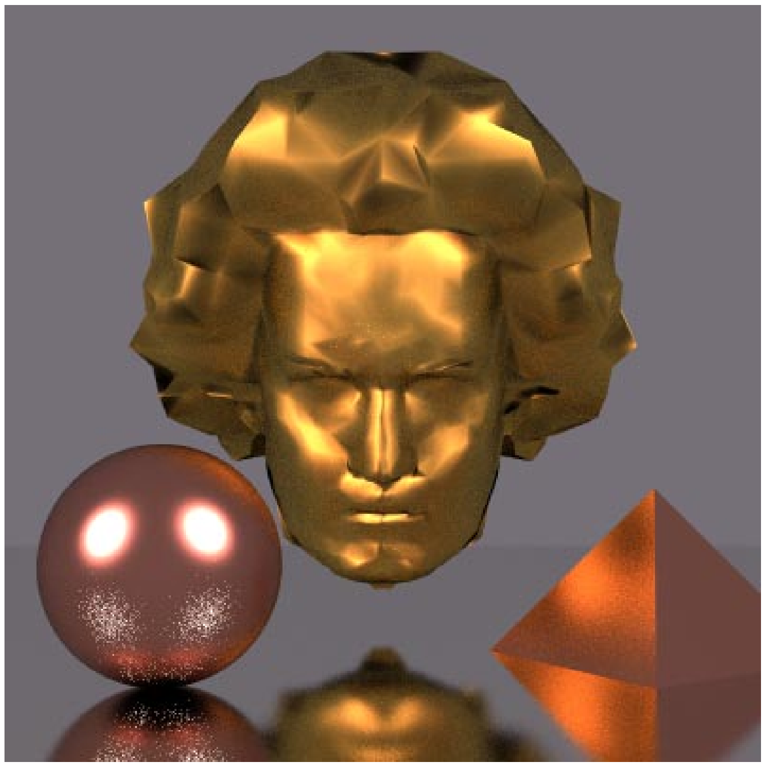


Figure 3.7: Metallic objects defined by the max-Phong model

As for the reciprocal Phong model, energy conservation requires that

$$k_s \leq \frac{n+2}{2\pi}.$$

At non-perpendicular illumination angles, only the cutting of the reflection lobe attenuates significantly the albedo, for shiny materials $\cos\theta / \max(\cos\theta, \cos\theta') \approx 1$ in the reflection lobe. Let us denote the factor representing the cutting of the reflection lobe by $\mathrm{Cut}^*(\theta')$. Note that $\mathrm{Cut}^*(\theta')$ is similar, but not identical to $\mathrm{Cut}(\theta')$.

$$a(\theta') = \mathrm{Cut}^*(\theta') \int_{\phi=0}^{2\pi} \int_{\psi=0}^{\pi/2} k_s \cdot \cos^n \psi \cdot \sin\psi \cdot \frac{\cos\theta}{\max(\cos\theta, \cos\theta')} \, d\psi d\phi \approx \mathrm{Cut}^*(\theta') \cdot \frac{2\pi}{n+1}.$$

If parameter k_s is defined from the Fresnel function, we have to specify which incident angle must be used. The angle between the surface normal and the light vector is not appropriate, because the BRDF will not be symmetric and the real normal vector where the reflection actually happens is not equal to the normal of the mean surface due to the surface irregularities (figure 3.8). This real normal vector is, in fact, a random variable. If the surface is a collection of randomly oriented ideal mirrors, then those surface elements which reflect from $\vec{L}$ to $\vec{V}$ has normal vector $\vec{H} = (\vec{L} + \vec{V})/2$. Thus the cosine of the incoming angle is computed from the $(\vec{H} \cdot \vec{L})$ scalar product for the Fresnel function.

3.4.2 Cook-Torrance model

Specular reflection can be more rigorously analyzed by modeling the surface irregularities by probability distributions, as has been proposed by Torrance, Sparrow, Cook and Blinn. In their model, the surface is assumed to consist of randomly oriented perfect mirrors, so-called **microfacets**. As in the previous section, the reflected light is broken down into diffuse and specular components. The diffuse component is believed to be generated by multiple reflections on the microfacets and also by emission of the absorbed light by the material of the surface. The diffuse component is well described by Lambert's law. The specular component, on the other hand, is produced by the direct reflections of the microfacets. This section discusses the derivation of the bi-directional transfer probability density $w(\vec{L}, \vec{V})$ for this specular part. Recall that the transfer function is the following probability density (equation (2.8)):

$$w(\vec{L}, \vec{V}) \cdot d\omega = \Pr\{\text{photon is reflected directly to } d\omega \text{ around } \vec{V} \mid \text{coming from } \vec{L}\}. \tag{3.15}$$

Concerning this type of reflection from direction $\vec{L}$ to $d\omega$ around direction $\vec{V}$, only those facets can contribute whose normal is in $d\omega_H$ around the halfway unit vector $\vec{H}$. If reflection is to happen, the facet must obviously be facing in the right direction. It should not be hidden by other facets, nor should its reflection run into other facets, and it should not absorb the photon for the possible contribution.

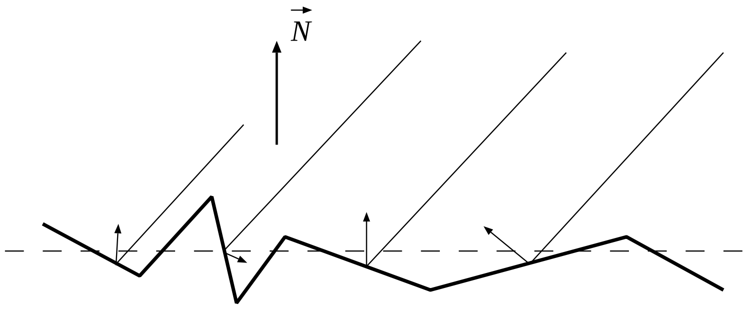


Figure 3.8: Microfacet model of the reflecting surface

Considering these facts, the event that "a photon is reflected directly to $d\omega$ around $\vec{V}$" can be expressed as the logical AND connection of the following events:

1. **Orientation**: In the path of the photon there is a microfacet having its normal in $d\omega_H$ around $\vec{H}$.
2. **No shadowing or masking**: The given microfacet is not hidden by other microfacets from the photon coming from the lightsource, and the reflected photon does not run into another microfacet.
3. **Reflection**: The photon is not absorbed by the microfacet that is supposed to be a perfect mirror.

Since these events are believed to be stochastically independent, their probability can be calculated independently, and the probability of the composed event will be their product.

Concerning the probability of the microfacet normal being in $d\omega_H$, we can suppose that all facets have equal area f. Let the probability density of the microfacet normal be $P(\vec{H})$. Blinn [Bli77] proposed **Gaussian distribution** for $P(\vec{H})$, since it seemed reasonable due to the central limit value theorem of probability theory:

$$P(\vec{H}) = \text{const} \cdot e^{-(\alpha/m)^2}, \tag{3.16}$$

where α is the angle of the microfacet with respect to the normal of the mean surface, that is the angle between $\vec{N}$ and $\vec{H}$, and m is the root mean square of the slope, i.e. a measure of the roughness.

Later Torrance and Sparrow showed that the results of the early work of Beckmann [BS63] and Davies [Dav54], who discussed the scattering of electromagnetic waves theoretically, can also be used here and thus Torrance proposed the **Beckmann distribution** function instead of the Gaussian:

$$P(\vec{H}) = \frac{1}{m^2 \cos^4 \alpha} \cdot e^{-\left(\frac{\tan^2 \alpha}{m^2}\right)}. \tag{3.17}$$

Let us examine a surface element dA. If a photon arrives from direction $\vec{L}$ to a surface element dA, the visible area of the surface element will be $dA \cdot (\vec{N} \cdot \vec{L})$, while the total visible area of the microfacets having their normal in the direction around $\vec{H}$ will be the product of the visible size of a single microfacet ($f \cdot (\vec{H} \cdot \vec{L})$), and the number of properly oriented microfacets. The number of properly oriented microfacets, in turn, is the product of the probability that a microfacet is properly oriented ($P(\vec{H}) \cdot d\omega_H$) and the number of all microfacets in dA (dA/f). The probability of finding an appropriate microfacet aligned with the photon is the visible size of properly oriented microfacets divided by the visible surface area, thus we obtain:

$$\Pr\{\text{orientation}\} = \frac{f \cdot (\vec{H} \cdot \vec{L}) \cdot P(\vec{H}) \cdot d\omega_H \cdot dA/f}{dA \cdot (\vec{N} \cdot \vec{L})} = \frac{P(\vec{H}) \cdot d\omega_H \cdot (\vec{H} \cdot \vec{L})}{(\vec{N} \cdot \vec{L})}. \tag{3.18}$$

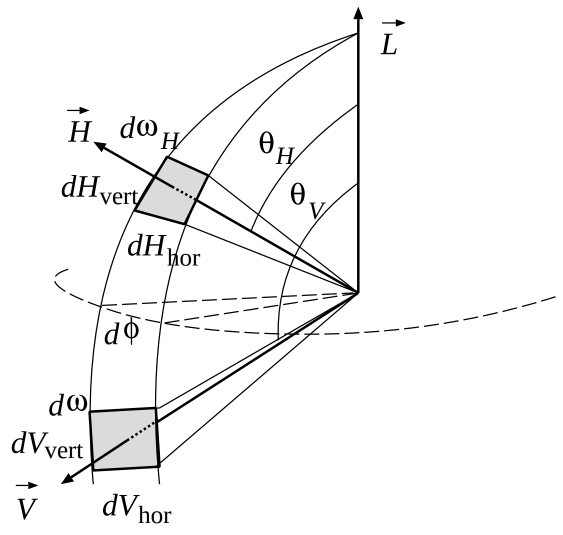


Figure 3.9: Calculation of $d\omega_H/d\omega$

Unfortunately, in the BRDF the transfer probability density uses $d\omega$ instead of $d\omega_H$, thus we have to determine $d\omega_H/d\omega$ [JGMHe88]. Defining a spherical coordinate system (θ, ϕ), with the north pole in the direction of $\vec{L}$ (figure 3.9), the solid angles are expressed by the product of vertical and horizontal arcs:

$$d\omega = dV_{\text{hor}} \cdot dV_{\text{vert}}, \quad d\omega_H = dH_{\text{hor}} \cdot dH_{\text{vert}}. \tag{3.19}$$

By using geometric considerations and applying the law of reflection, we get:

$$dV_{\text{hor}} = d\phi \cdot \sin\theta_V, \quad dH_{\text{hor}} = d\phi \cdot \sin\theta_H, \quad dV_{\text{vert}} = 2dH_{\text{vert}}. \tag{3.20}$$

This in turn yields:

$$\frac{d\omega_H}{d\omega} = \frac{\sin\theta_H}{2\sin\theta_V} = \frac{\sin\theta_H}{2\sin 2\theta_H} = \frac{1}{4\cos\theta_H} = \frac{1}{4(\vec{L}\cdot\vec{H})}. \tag{3.21}$$

since $\theta_V = 2 \cdot \theta_H$. Thus the orientation probability is

$$\Pr\{\text{orientation}\} = \frac{P(\vec{H}) \cdot (\vec{H} \cdot \vec{L})}{(\vec{N} \cdot \vec{L})} \cdot \frac{d\omega_H}{d\omega} \cdot d\omega = \frac{P(\vec{H})}{4(\vec{N} \cdot \vec{L})} \cdot d\omega. \tag{3.22}$$

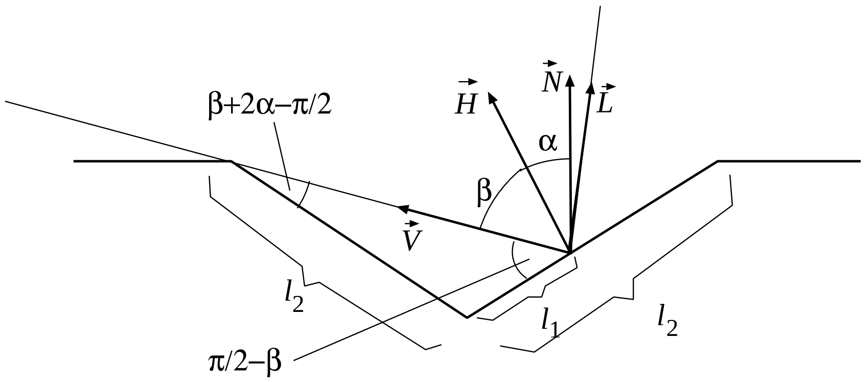


Figure 3.10: Geometry of masking

The visibility of the microfacets from direction $\vec{V}$ means that the reflected photon does not run into another microfacet. The collision is often referred to as **masking**. Looking at figure 3.10, we can easily recognize that the probability of masking is l_1/l_2, where l_2 is the one-dimensional length of the microfacet, and l_1 describes the boundary case from where the beam is masked. The angles of the triangle formed by the bottom of the microfacet wedge and the beam in the boundary case can be expressed by the angles $\alpha = \text{angle}(\vec{N}, \vec{H})$ and $\beta = \text{angle}(\vec{V}, \vec{H}) = \text{angle}(\vec{L}, \vec{H})$ by geometric considerations and by using the law of reflection. Applying the sine law for this triangle, and some trigonometric formulae:

$$\Pr\{\text{not masking}\} = 1 - \frac{l_1}{l_2} = 1 - \frac{\sin(\beta + 2\alpha - \pi/2)}{\sin(\pi/2 - \beta)} = 2 \cdot \frac{\cos\alpha \cdot \cos(\beta + \alpha)}{\cos\beta}. \tag{3.23}$$

According to the definitions of the angles $\cos\alpha = \vec{N} \cdot \vec{H}$, $\cos(\beta + \alpha) = \vec{N} \cdot \vec{V}$ and $\cos\beta = \vec{V} \cdot \vec{H}$.

If the angle of incident light and the facet normal do not allow the triangle to be formed, the probability of no masking taking place is obviously 1. This situation can be recognized by evaluating the formula without any previous considerations and checking whether the result is greater than 1, then limiting the result to 1. The final result is:

$$\Pr\{\text{not masking}\} = \min\{2 \cdot \frac{(\vec{N} \cdot \vec{H}) \cdot (\vec{N} \cdot \vec{V})}{(\vec{V} \cdot \vec{H})}, 1\}. \tag{3.24}$$

The probability of **shadowing** can be derived in exactly the same way, only $\vec{L}$ should be substituted for $\vec{V}$:

$$\Pr\{\text{not shadowing}\} = \min\{2 \cdot \frac{(\vec{N} \cdot \vec{H}) \cdot (\vec{N} \cdot \vec{L})}{(\vec{L} \cdot \vec{H})}, 1\}. \tag{3.25}$$

The probability of neither shadowing nor masking taking place can be approximated by the minimum of the two probabilities:

$$\Pr\{\text{no shadow and mask}\} \approx \min\{2 \cdot \frac{(\vec{N} \cdot \vec{H}) \cdot (\vec{N} \cdot \vec{V})}{(\vec{V} \cdot \vec{H})}, 2 \cdot \frac{(\vec{N} \cdot \vec{H}) \cdot (\vec{N} \cdot \vec{L})}{(\vec{L} \cdot \vec{H})}, 1\} = G(\vec{N}, \vec{L}, \vec{V}). \tag{3.26}$$

Note that this approximation upperbounds the real probability, particularly at grazing angles, which can result in the violation of the energy balance.

When analyzing the perfect mirrors, we concluded that even a perfect mirror absorb some portion of the incident light, as is described by the **Fresnel equations**. Since $F(\lambda, \theta')$ is the fraction of the reflected energy, it also describes the probability of a photon being reflected at a microfacet that is assumed to be an ideal mirror, giving:

$$\Pr\{\text{reflection}\} = F(\lambda, \vec{H} \cdot \vec{L})$$

where variable θ' has been replaced by $\vec{H} \cdot \vec{L}$ since those microfacets that reflect from $\vec{L}$ to $\vec{V}$ have normal vector equal to $\vec{H}$.

Now we can summarize the results by multiplying the probabilities of the independent events to express $w(\vec{L},\vec{V})d\omega$:

$$w(\vec{L},\vec{V})d\omega = \mathrm{Pr}\{\text{orientation}\} \cdot \mathrm{Pr}\{\text{no mask and shadow}\} \cdot \mathrm{Pr}\{\text{reflection}\} =$$

$$\frac{P(\vec{H})}{4(\vec{N}\cdot\vec{L})} \cdot G(\vec{N},\vec{L},\vec{V}) \cdot F(\lambda, \vec{H}\cdot\vec{L})\, d\omega. \tag{3.27}$$

The BRDF is the transfer probability divided by $d\omega$ and the cosine of the outgoing angle $(\vec{N}\cdot\vec{V})$, thus we obtain

$$f_{r,\mathrm{Cook-Torrance}}(\vec{L},\vec{V}) = \frac{P(\vec{H})}{4(\vec{N}\cdot\vec{L})(\vec{N}\cdot\vec{V})} \cdot G(\vec{N},\vec{L},\vec{V}) \cdot F(\lambda, \vec{H}\cdot\vec{L}). \tag{3.28}$$

Chapter 4

Solution strategies for the global illumination problem

Since the rendering or the potential equations contain the unknown radiance function both inside and outside the integral, in order to express the solution, this coupling should be resolved. The possible solution techniques fall into one of the following three categories: **inversion**, **expansion** and **iteration**.

Operator $\mathcal{T}$ represents light-surface interaction, thus each of its application generates a higher-bounce estimate of the light transport (or alternatively $\mathcal{T}'$ represents potential-surface interaction). For physically plausible optical material models, a reflection or refraction always decreases the total energy, thus the integral operator is always a contraction. However, when the transport is evaluated numerically, computation errors may pose instability problems if the scene is highly reflective. As we shall see, expansion and iteration exploit the contractive property of the transport operator, but inversion does not.

4.1 Inversion

Inversion groups the terms that contain the unknown function on the same side of the equation and applies formally an inversion operation:

$$(1-\mathcal{T})L = L^e \implies L = (1-\mathcal{T})^{-1}L^e. \tag{4.1}$$

Thus the measured power is

$$\mathcal{M}L = \mathcal{M}(1-\mathcal{T})^{-1}L^e. \tag{4.2}$$

However, since $\mathcal{T}$ is infinite dimensional, it cannot be inverted in closed form. Thus it should be approximated by a finite dimensional mapping, that is usually given as a matrix. This kind of approximation is provided by **finite-element method**s that project the problem into a finite dimensional function space, and approximate the solution here. This projection converts the original integral equation to a system of linear equations, which can be inverted, for example, by Gaussian elimination method. This approach was used in early radiosity methods, but have been dropped due to the cubic time complexity and the numerical instability of the matrix inversion.

Inversion has a unique property that is missing in the other two methods. Its efficiency does not depend on the contractivity of the integral operator, neither does it even require the integral operator to be a contraction.

4.2 Expansion

Expansion techniques eliminate the coupling by obtaining the solution in the form of an infinite Neumann series.

4.2.1 Expansion of the rendering equation: gathering walks

Substituting the right side's L by $L^e + \mathcal{T}L$, which is obviously L according to the equation, we get:

$$L = L^e + \mathcal{T}L = L^e + \mathcal{T}(L^e + \mathcal{T}L) = L^e + \mathcal{T}L^e + \mathcal{T}^2 L. \tag{4.3}$$

Repeating this step n times, the original equation can be expanded into a Neumann series:

$$L = \sum_{i=0}^{n} \mathcal{T}^i L^e + \mathcal{T}^{n+1} L. \tag{4.4}$$

If integral operator $\mathcal{T}$ is a contraction, then $\lim_{n\to\infty} \mathcal{T}^{n+1}L = 0$, thus

$$L = \sum_{i=0}^{\infty} \mathcal{T}^i L^e. \tag{4.5}$$

The measured power is

$$\mathcal{M}L = \sum_{i=0}^{\infty} \mathcal{M}\mathcal{T}^i L^e. \tag{4.6}$$

The terms of this infinite Neumann series have intuitive meaning as well: $\mathcal{M}\mathcal{T}^0 L^e = \mathcal{M}L^e$ comes from the emission, $\mathcal{M}\mathcal{T}^1 L^e$ comes from a single reflection, $\mathcal{M}\mathcal{T}^2 L^e$ from two reflections, etc.

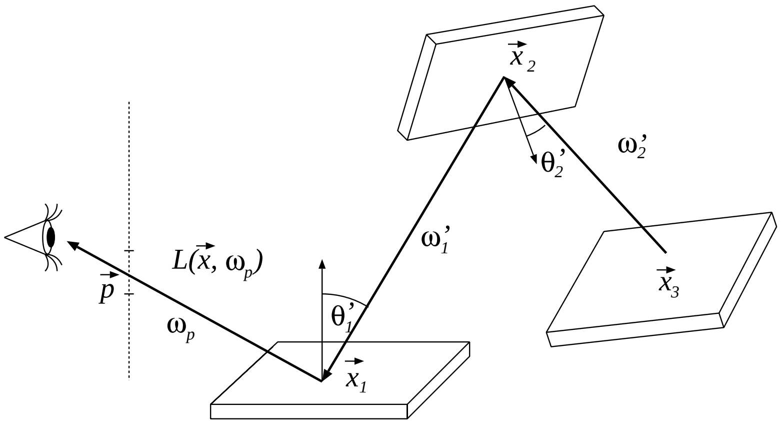


Figure 4.1: The integrand of $\mathcal{M}\mathcal{T}^2 L^e$ is a two-step gathering walk

In order to understand how this can be used to determine the power going through a single pixel, let us examine the structure of $\mathcal{M}\mathcal{T}^i L^e$ as a single multi-dimensional integral for the $i = 2$ case:

$$\mathcal{M}(\mathcal{T}^2 L^e) = \int\limits_{S_p} \int\limits_{\Omega_1'} \int\limits_{\Omega_2'} \frac{c(\vec{p})}{S_p} \cdot w_1(\vec{x}_1) \cdot w_2(\vec{x}_2) \cdot L^e(\vec{x}_3, \omega_2')\, d\omega_2' d\omega_1' d\vec{p}. \tag{4.7}$$

where S_p is the pixel area, $c(\vec{p})$ is the camera parameter, $\vec{p}$ is a running point on this pixel, ω_1' and ω_2' are the directions of the first and second reflections, respectively (figure 4.1) and

$$\begin{aligned} \vec{x}_1 &= h(\vec{p}, -\omega_{\vec{p}}), \\ \vec{x}_2 &= h(\vec{x}_1, -\omega_1'), \\ \vec{x}_3 &= h(\vec{x}_2, -\omega_2') = h(h(\vec{x}_1, -\omega_1'), -\omega_2'), \end{aligned} \tag{4.8}$$

and the weights are

$$\begin{aligned} w_1 &= f_r(\omega_1', \vec{x}_1, \omega_{\vec{p}}) \cdot \cos\theta_1', \\ w_2 &= f_r(\omega_2', \vec{x}_2, \omega_1') \cdot \cos\theta_2'. \end{aligned} \tag{4.9}$$

Thus to evaluate the integrand at point $(\vec{p}, \omega_1', \omega_2')$, the following algorithm must be executed:

1. Point $\vec{x}_1 = h(\vec{p}, -\omega_{\vec{p}})$ that is visible through the point $\vec{p}$ of the pixel from the eye should be found. This can be done by sending a ray from the eye into the direction of $\vec{p}$ and identifying the surface that is first intersected.

2. Surface point $\vec{x}_2 = h(\vec{x}_1, -\omega_1')$ — that is the point which is visible from $\vec{x}_1$ at direction $-\omega_1'$ — must be determined. This can be done by sending another ray from $\vec{x}_1$ into direction $-\omega_1'$ and identifying the surface that is first intersected.

3. Surface point $\vec{x}_3 = h(h(\vec{x}_1, -\omega_1'), -\omega_2')$ — that is the point visible from $\vec{x}_2$ at direction $-\omega_2'$ — is identified. This means the continuation of the ray tracing at direction $-\omega_2'$.

4. The emission intensity of the surface at $\vec{x}_3$ in the direction of ω_2' is obtained and is multiplied with the cosine terms and the BRDFs of the two reflections.

This algorithm can easily be generalized for arbitrary number of reflections. A ray is emanated recursively from the visible point at direction ω'_1 then from the found surface at ω'_2, etc. until ω'_n. The emission intensity at the end of the walk is read and multiplied by the BRDFs and the cosine terms of the stages of the walk. These walks provide the value of the integrand at "point" $\vec{p}, \omega'_1, \omega'_2, \ldots, \omega'_n$. Note that a single walk of length n can be used to estimate the 1-bounce, 2-bounce, etc. n-bounce transfer simultaneously, if the emission is transferred not only from the last visited point but from all visited points.

The presented walking technique starts at the eye and **gathers** the illumination encountered during the walk. The gathered illumination is attenuated according to the cosine weighted BRDFs of the path.

So far, we have examined the structure of the terms of the Neumann series as a single multi-dimensional integral. Alternatively, it can also be considered as recursively evaluating many directional integrals. For example, the two-bounce transfer is:

$$\mathcal{M}\mathcal{T}^2 L^e = \int\limits_{S_p} w_0 \cdot \left[\int\limits_{\Omega'_1} w_1 \cdot \left[\int\limits_{\Omega'_2} w_2 \cdot L^e \, d\omega'_2 \right] d\omega'_1 \right] d\vec{p}. \tag{4.10}$$

In order to estimate the outer integral of variable $\vec{p}$, the integrand is needed in sample point $\vec{p}$. This, in turn, requires the estimation of the integral of variable ω'_1 at $\vec{p}$, which recursively needs again the approximation of the integral of variable ω'_2 at $(\vec{p}, \omega'_1)$.

If the same number — say m — of sample points are used for each integral quadrature, then this recursive approach will use m points for the 1-bounce transfer, m^2 for the two-bounces, m^3 for the three-bounces, etc. This kind of subdivision of paths is called **splitting** [AK90]. Splitting becomes prohibitive for high-order reflections and is not even worth doing because of the contractive property of the integral operator. Due to the contraction, the contribution of higher-order bounces is less, thus it is not very wise to compute them as accurately as low-order bounces.

4.2.2 Expansion of the potential equation: shooting walks

The potential equation can also be expanded into a Neumann series similarly to the rendering equation:

$$W = \sum_{i=0}^{\infty} \mathcal{T}'^i W^e, \tag{4.11}$$

which results in the following measured power

$$\mathcal{M}'W = \sum_{i=0}^{\infty} \mathcal{M}'\mathcal{T}'^i W^e. \tag{4.12}$$

$\mathcal{M}'W^e$ is the power measured by the device from direct emission, $\mathcal{M}'\mathcal{T}'W^e$ is the power after a single reflection, $\mathcal{M}'\mathcal{T}'^2 W^e$ is after two reflections, etc.

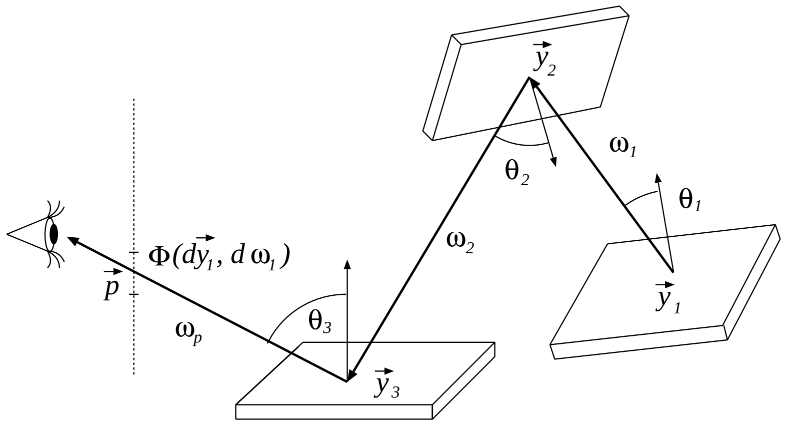


Figure 4.2: The integrand of $\mathcal{M}'\mathcal{T}'^2 W^e$ is a two-step shooting walk

Let us again consider the structure of $\mathcal{M}'\mathcal{T}'^2 W^e$:

$$\mathcal{M}'\mathcal{T}'^2 W^e = \int\limits_{S} \int\limits_{\Omega_1} \int\limits_{\Omega_2} \int\limits_{\Omega_3} L^e(\vec{y}_1, \omega_1) \cdot w_0(\vec{y}_1) \cdot w_1(\vec{y}_2) \cdot w_2(\vec{y}_3) \cdot W^e(\vec{y}_3, \omega_3) \, d\omega_3 d\omega_2 d\omega_1 d\vec{y}_1, \tag{4.13}$$

where

$$\vec{y}_2 = h(\vec{y}_1, \omega_1),$$
$$\vec{y}_3 = h(\vec{y}_2, \omega_2) = h(h(\vec{y}_1, \omega_1), \omega_2) \tag{4.14}$$

and the weights are

$$w_0 = \cos\theta_1,$$
$$w_1 = f_r(\omega_1, \vec{y}_2, \omega_2) \cdot \cos\theta_2,$$
$$w_2 = f_r(\omega_2, \vec{y}_3, \omega_3) \cdot \cos\theta_3. \tag{4.15}$$

Substituting the measuring function of the pin-hole camera (equation (2.45)), we can conclude that $\mathcal{M}'\mathcal{T}'^2 W^e$ can be non-zero only if $\vec{y}_3$ is visible through the given pixel, and the integral over Ω_3 is simplified to a single value by the Dirac-delta function, thus the final form of the two-bounce reflection is:

$$\int_S \int_{\Omega_1} \int_{\Omega_2} L^e(\vec{y}_1, \omega_1) \cdot w_0(\vec{y}_1) \cdot w_1(\vec{y}_2) \cdot w_2(\vec{y}_3) \cdot g(\vec{y}_3)\, d\omega_2 d\omega_1 d\vec{y}_1. \tag{4.16}$$

if $\vec{y}_3$ is visible though the given pixel and 0 otherwise, where $g(\vec{y}_3)$ is the surface dependent camera parameter.

Thus to evaluate the integrand at point $(\vec{y}_1, \omega_1, \omega_2)$, the following algorithm must be executed:

1. The cosine weighted emission of point $\vec{y}_1$ in direction ω_1 is computed. Surface point $\vec{y}_2 = h(\vec{y}_1, \omega_1)$ — that is the point which is visible from $\vec{y}_1$ at direction ω_1 — must be determined. This can be done by sending a ray from $\vec{y}_1$ into direction ω_1 and identifying the surface that is first intersected. This point "receives" the computed cosine weighted emission.

2. Surface point $\vec{y}_3 = h(h(\vec{y}_1, \omega_1), \omega_2)$ — that is the point visible from $\vec{y}_2$ at direction ω_2 — is identified. This means the continuation of the ray tracing at direction ω_2. The emission is weighted again by the local BRDF and the cosine of the normal and the outgoing direction.

3. It is determined whether or not this point $\vec{y}_3$ is visible from the eye, and through which pixel. The contribution to this pixel is obtained by weighting the transferred emission by the local BRDF, cosine angle and the surface dependent camera parameter.

This type of walk, called **shooting**, starts at a known point $\vec{y}_1$ of a lightsource and simulates the photon reflection for a few times and finally arrives at a pixel whose radiance this walk contributes to.

Note that in gathering walks the BRDF is multiplied with the cosine of the angle between the normal and the incoming direction, while in shooting walks with the cosine of the angle between the normal and the outgoing direction.

4.2.3 Merits and disadvantages of expansion methods

The main problem of expansion techniques is that they require the evaluation of very high dimensional integrals that appear as terms in the infinite series. Practical implementations usually truncate the infinite Neumann series, which introduces some bias, or stop the walks randomly, which significantly reduces the samples of higher order interreflections. These can result in visible artifacts for highly reflective scenes.

On the other hand, expansion methods also have an important advantage. Namely, they do not require temporary representations of the complete radiance function, thus do not necessitate finite-element approximations. Consequently, these algorithms can work with the original geometry without tessellating the surfaces to planar polygons.

Expansion techniques generate random walks independently. It can be an advantage, since these algorithms are suitable for **parallel computing**. However, it also means that these methods "forget" the previous history of walks, and they cannot reuse the visibility information gathered when computing the previous walks, thus they are not as fast as they could be.

4.3 Iteration

Iteration techniques realize that the solution of the rendering equation is the fixed point of the following iteration scheme

$$L_n = L^e + \mathcal{T} L_{n-1}, \tag{4.17}$$

thus if operator $\mathcal{T}$ is a contraction, then this scheme will converge to the solution from any initial function L_0. The measured power can be obtained as a limiting value

$$\mathcal{M}L = \lim_{n \to \infty} \mathcal{M}L_n. \tag{4.18}$$

In order to store the approximating functions L_n, usually **finite-element method**s are applied, as for example, in **diffuse radiosity** [SC94], or in non-diffuse radiosity using **partitioned hemisphere** [ICG86], **directional distributions** [SAWG91] or **illumination networks** [BF89].

There are two critical problems here. On the one hand, since the domain of L_n is 4 dimensional and L_n has usually high variation, an accurate finite-element approximation requires very many basis functions, which, in turn, need a lot of storage space. Although **hierarchical methods** [HSA91, AH93], **wavelet** or **multiresolution methods** [CSSD96, SGCH94] and **clustering** [SDS95, CLSS97, SPS98] can help, the memory requirements are still prohibitive for complex scenes. This problem is less painful for the diffuse case since here the domain of the radiance is only 2 dimensional.

On the other hand, when finite element techniques are applied, operator $\mathcal{T}$ is only approximated, which introduces some non-negligible error in each step. If the contraction ratio of the operator is λ, then the total accumulated error will be approximately $1/(1-\lambda)$ times the error of a single step [SKFNC97]. For highly reflective scenes — i.e. when λ is close to 1 — the iteration is slow and the result is inaccurate if the approximation of the operator is not very precise. Very accurate approximations of the transport operator, however, require a lot of computation time and storage space.

In the diffuse radiosity setting several methods have been proposed to improve the quality of the iteration. For example, we can use Chebyshev iteration instead of the Jacobi or the Gauss-Seidel method for such ill conditioned systems [BBS96]. On the other hand, realizing that the crucial part of designing such an the algorithm is finding a good and "small" approximation of the transport operator, the method called **well-distributed ray-sets** [NNB97, BNN$^+$98] proposes the adaptive approximation of the transport operator. This approximation is a set of rays selected carefully taking into account the important patches and directions. In [BNN$^+$98], the adaptation of the discrete transport operator is extended to include patch subdivision as well, to incorporate the concepts of **hierarchical radiosity** [HSA91]. The adaptation strategy is to refine the discrete approximation (by adding more rays to the set), when the iteration with the coarse approximation is already stabilized. Since the discrete approximation of the transport operator is not constant but gets finer in subsequent phases, the error accumulation problem can be controlled but is not eliminated.

This thesis proposes a new method called the **stochastic iteration** to attack both the problem of prohibitive memory requirements and the problem of error accumulation.

Compared to expansion techniques, iteration has both advantages and disadvantages. Its important advantage is that it can potentially reuse all the information gained in previous computation steps and can exploit the coherence of the radiance function, thus iteration is expected to be faster than expansion. Iteration can also be seen as a single infinite length random walk. If implemented carefully, iteration does not reduce the number of estimates for higher order interreflections, thus it is more robust when rendering highly reflective scenes than expansion.

The property that iteration requires tessellation and finite-element representation is usually considered as a disadvantage. And indeed, sharp shadows and highlights on highly specular materials can be incorrectly rendered and light-leaks may appear, not to mention the unnecessary increase of the complexity of the scene description (think about, for example, the definition of an original and a tessellated sphere). However, finite-element representation can also provide smoothing during all stages of rendering, which results in more visually pleasing and dot-noise free images. Summarizing, iteration is the better option if the scene is not highly specular.

4.3.1 Analysis of the iteration

In order to find necessary conditions for the convergence, let us examine two subsequent iteration steps:

$$\begin{aligned} L_n &= L^e + \mathcal{T}L_{n-1}, \\ L_{n-1} &= L^e + \mathcal{T}L_{n-2}. \end{aligned} \tag{4.19}$$

Substracting the two equations and assuming that $L_0 = 0$, we can obtain:

$$L_n - L_{n-1} = \mathcal{T}(L_{n-1} - L_{n-2}) = \mathcal{T}^{n-1}(L_1 - L_0) = \mathcal{T}^{n-1}L^e. \tag{4.20}$$

If operator $\mathcal{T}$ is a contraction, that is if

$$||\mathcal{T}L|| < \lambda \cdot ||L||, \qquad \lambda < 1, \tag{4.21}$$

with some function norm, then

$$||L_n - L_{n-1}|| = ||\mathcal{T}^{n-1} L^e|| < \lambda^{n-1} \cdot ||L^e||. \tag{4.22}$$

Thus iteration converges to the real solution at least with the speed of a geometric series. The contraction ratio λ depends on both the geometry and the average reflectivity of the scene. From the point of view of the geometry, the contraction ratio is maximum if the environment is closed, when λ corresponds to the average albedo.

Error caused by the approximation of the transport operator

In practice operator $\mathcal{T}$ cannot be evaluated exactly, which introduces some error in each step of the iteration. The errors may accumulate in the final solution. This section examines this phenomenon.

Assume that an operator $\mathcal{T}^*$ which is only an approximation of $\mathcal{T}$ is used in the iteration. Suppose that both operators are contractions, thus both iterations will converge from any initial function.

Let the radiance after n iterations of operator $\mathcal{T}^*$ be L_n and let us assume that the iteration starts at the solution of the exact equation, that is $L_0 = L$ (since the iteration converges to the same limit from any initial function, this assumption can be made). The solution of the approximated equation is $L^* = L_\infty$. The error at step n is

$$||L_n - L|| = ||L_n - L_{n-1} + L_{n-1} - ... + L_1 - L|| \leq \sum_{i=1}^{n} ||L_i - L_{i-1}||. \tag{4.23}$$

Since if $i > 1$, then

$$||L_i - L_{i-1}|| = ||\mathcal{T}^* L_{i-1} - \mathcal{T}^* L_{i-2}|| = ||\mathcal{T}^*(L_{i-1} - L_{i-2})|| \leq \lambda \cdot ||L_{i-1} - L_{i-2}|| \leq \lambda^{i-1} \cdot ||L_1 - L_0||$$

we have

$$\sum_{i=1}^{n} ||L_i - L_{i-1}|| \leq ||L_1 - L_0|| \cdot (1 + \lambda + \lambda^2 + \ldots \lambda^{n-1}). \tag{4.24}$$

Letting n go to infinity, we obtain the error between the fixed points of the original and the approximated equations

$$||L^* - L|| \leq ||L_1 - L_0|| \cdot (1 + \lambda + \lambda^2 + \ldots) = \frac{||L_1 - L_0||}{1 - \lambda}. \tag{4.25}$$

On the other hand, subtracting the real solution defined as the fixed point of the exact equation $L = L^e + \mathcal{T} L$ from the first iteration of the approximated operator

$$L_1 = L^e + \mathcal{T}^* L,$$

we obtain

$$L_1 - L = \mathcal{T}^* L - \mathcal{T} L. \tag{4.26}$$

Thus the final error formula is

$$||L^* - L|| \leq \frac{||\mathcal{T}^* L - \mathcal{T} L||}{1 - \lambda}. \tag{4.27}$$

4.4 Analytical solution of the rendering equation

In order to test the error and convergence of global illumination algorithms, we need scenes for which the exact solution is known. Generally, there are two alternatives. Either we use simple scenes for which analytical solution is possible, or numerical methods are applied using so many samples that the approximate solution can be accepted as a reference.

In order to find analytically solvable scenes, we use a reverse approach. Normally, a scene is analyzed to find the radiance distribution. Now, we start with a prescribed radiance distribution and search for scenes where the radiance would be identical to the given radiance. Two special cases are examined. In the first case we are looking for scenes where the radiance is constant everywhere and at every direction. In the second case we establish criteria for making only the reflected radiance constant.

4.4.1 Scenes with constant radiance

Suppose that the constant radiance is $\tilde{L}$ and also that the scene is a **closed environment**, i.e. looking at any direction we can see a surface, thus the incoming radiance is also constant. Substituting this to the rendering equation we get:

$$\tilde{L} = L^e(\vec{x},\omega) + (\mathcal{T}\tilde{L})(\vec{x},\omega) = L^e(\vec{x},\omega) + \tilde{L}\cdot(\mathcal{T}1)(\vec{x},\omega) = L^e(\vec{x},\omega) + \tilde{L}\cdot a(\vec{x},\omega), \tag{4.28}$$

since the albedo is the response to homogeneous unit illumination. According to this equation, the radiance will be constant for scenes of arbitrary geometry and of arbitrary emission function if the albedo function is:

$$a(\vec{x},\omega) = 1 - \frac{L^e(\vec{x},\omega)}{\tilde{L}}. \tag{4.29}$$

If L^e is constant, then the required albedo will also be constant. In this special case — called the homogeneous diffuse environment — the geometry is arbitrary, but all surfaces have the same diffuse reflection and emission [Shi91b, Kel95].

4.4.2 Scenes with constant reflected radiance

Let us assume that the reflected radiance — i.e. the radiance without the emission — is $\tilde{L}_r$ and assume again that the scene is closed. Substituting this into the rendering equation, we obtain:

$$L(\vec{x},\omega) = L^e(\vec{x},\omega) + \tilde{L}_r = L^e(\vec{x},\omega) + (\mathcal{T}(L^e+\tilde{L}_r))(\vec{x},\omega) = L^e(\vec{x},\omega) + (\mathcal{T}L^e)(\vec{x},\omega) + \tilde{L}_r\cdot a(\vec{x},\omega). \tag{4.30}$$

This imposes the following constraint on the albedo function:

$$a(\vec{x},\omega) = 1 - \frac{(\mathcal{T}L^e)(\vec{x},\omega)}{\tilde{L}_r}. \tag{4.31}$$

Note that if the reflected radiance is constant, then the albedo is also constant, which is guaranteed if the BRDF is diffuse and has the same f_r value everywhere. An appropriate geometry, which makes $\mathcal{T}L^e$ constant for arbitrary diffuse emission if the BRDF is diffuse and constant, is the internal surface of the sphere as proposed originally by [HMF98]. Formally, let the scene be an inner surface S of a sphere of radius R, the BRDF be constant $f_r = a/\pi$ and the emission be diffuse and defined by $L^e(\vec{x})$. Using the

$$d\omega' = \frac{\cos\theta_{\vec{y}}\cdot d\vec{y}}{|\vec{y}-\vec{x}|^2}$$

substitution for the solid angle (equation (2.2)), we obtain the following form of the light transport operator:

$$(\mathcal{T}L^e)(\vec{x},\omega) = \int_S f_r\cdot\cos\theta_{\vec{x}}\cdot L^e(\vec{y})\cdot\frac{\cos\theta_{\vec{y}}}{|\vec{y}-\vec{x}|^2}\,d\vec{y}. \tag{4.32}$$

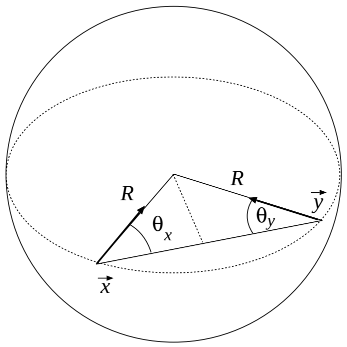


Figure 4.3: Geometry of the reference scene

Looking at figure 4.3, we can see that inside a sphere $\cos\theta_{\vec{x}} = \cos\theta_{\vec{y}} = \frac{|\vec{y}-\vec{x}|}{2R}$, thus we can obtain the following final form:

$$(\mathcal{T}L^e)(\vec{x},\omega) = \int_S f_r\cdot L^e(\vec{y})\cdot\frac{\cos\theta_{\vec{x}}\cdot\cos\theta_{\vec{y}}}{|\vec{y}-\vec{x}|^2}\,d\vec{y} = \frac{f_r}{4R^2}\cdot\int_S L^e(\vec{y})\,d\vec{y} = a\cdot\frac{\int_S L^e(\vec{y})\,d\vec{y}}{4R^2\pi}. \tag{4.33}$$

The response is equal to the product of the albedo and the average emission of the total spherical surface.

Chapter 5

Finite-element methods for the Global Illumination Problem

Iteration requires the representation of the temporary radiance function L_n. So does expansion if view-independent solution is needed since the final radiance distribution must be represented in a continuous domain. The exact representation of such functions might need infinite data, which is not available for practical algorithms. Thus finite approximations are required.

To represent a function over a continuous domain, **finite-element method**s can be used which approximate the function in a finite function series form:

$$L(\vec{x},\omega) \approx L^{(n)}(\vec{x},\omega) = \sum_{j=1}^{n} \mathbf{L}_j \cdot b_j(\vec{x},\omega) = \mathbf{b}^T(\vec{x},\omega) \cdot \mathbf{L} \tag{5.1}$$

where $b_j(\vec{x},\omega)$ is a system of predefined basis functions, and $\mathbf{L}_j$ factors are unknown coefficients.

This representation can also be seen as projecting the infinite dimensional space of the possible radiance functions into a finite-dimensional function space defined by the basis functions.

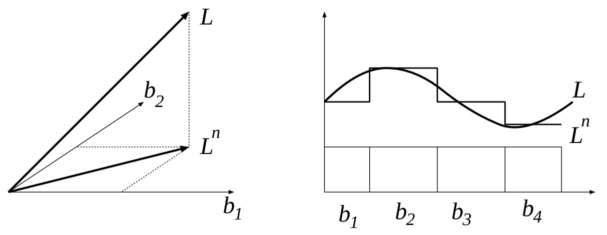


Figure 5.1: Finite element approximation

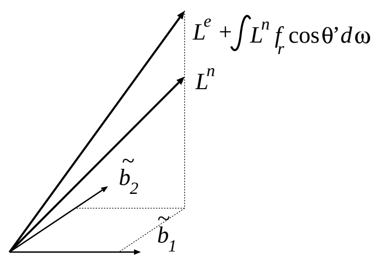


Figure 5.2: Projection to the adjoint base

Substituting this approximation into the rendering equation we can obtain:

$$\sum_{j=1}^{n} \mathbf{L}_j \cdot b_j(\vec{x},\omega) \approx \sum_{j=1}^{n} \mathbf{L}^e_j \cdot b_j(\vec{x},\omega) + \mathcal{T} \sum_{j=1}^{n} \mathbf{L}_j \cdot b_j(\vec{x},\omega) = \sum_{j=1}^{n} \mathbf{L}^e_j \cdot b_j(\vec{x},\omega) + \sum_{j=1}^{n} \mathbf{L}_j \cdot \mathcal{T} b_j(\vec{x},\omega). \tag{5.2}$$

Note that equality cannot be guaranteed, since even if $\sum_{j=1}^{n} \mathbf{L}_j \cdot b_j(\vec{x}, \omega)$ is in the subspace defined by the basis functions, the integral operator $\mathcal{T}$ may result in a function that is out of this space. Instead, equality is required in an appropriate subspace defined by **adjoint basis** functions $\tilde{b}_1(\vec{x}), \tilde{b}_2(\vec{x}), \ldots \tilde{b}_n(\vec{x})$ (figure 5.2). This set is required to be orthogonal to the original base $b_1(\vec{x}), b_2(\vec{x}), \ldots b_n(\vec{x})$ in the following sense:

$$\langle b_i(\vec{x}), \tilde{b}_j(\vec{x}) \rangle = \begin{cases} 1 \text{ if } i = j, \\ \\ 0 \text{ otherwise.} \end{cases} \tag{5.3}$$

Having projected equation 5.2 to the adjoint base — i.e. multiplying it by each adjoint basis functions $\tilde{b}_i$ — we obtain the following system of linear equations:

$$\mathbf{L}_i = \mathbf{L}_i^e + \sum_j \langle \mathcal{T} b_j, b_i \rangle \cdot \mathbf{L}_j. \tag{5.4}$$

This system of linear equations can also be presented in a vector form:

$$\mathbf{L} = \mathbf{L}^e + \mathbf{R} \cdot \mathbf{L}, \qquad \mathbf{R}_{ij} = \langle \mathcal{T} b_j, \tilde{b}_i \rangle. \tag{5.5}$$

An adjoint of this linear equation can be derived by selecting the adjoint base as the normalization of the measurement functions. Suppose that each basis function b_i is associated with a measurement device W_i^e that measures the power $\mathbf{P}_i$ leaving the support of the basis function, thus the appropriate adjoint basis function is $\tilde{b}_i = W_i^e / \langle b_i, W_i^e \rangle$. Thus the measured radiance is

$$\langle \sum_{j=1}^{n} \mathbf{L}_j \cdot b_j, W_i^e \rangle = \langle b_i, W_i^e \rangle \cdot \mathbf{L}_i = \mathbf{P}_i.$$

Similarly, the measured emission is

$$\langle \sum_{j=1}^{n} \mathbf{L}_j^e \cdot b_j, W_i^e \rangle = \langle b_i, W_i^e \rangle \cdot \mathbf{L}_i^e = \mathbf{P}_i^e.$$

Applying measurement operator W_i^e for equation (5.2), we can obtain the following equation:

$$\langle b_i, W_i^e \rangle \cdot \mathbf{L}_i = \langle b_i, W_i^e \rangle \cdot \mathbf{L}_i^e + \sum_{j=1}^{n} \langle \mathcal{T} b_j, W_i^e \rangle \cdot \mathbf{L}_j. \tag{5.6}$$

Replacing the radiances by measured values, we get

$$\mathbf{P}_i = \mathbf{P}_i^e + \sum_{j=1}^{n} \frac{\langle \mathcal{T} b_j, W_i^e \rangle}{\langle b_j, W_j^e \rangle} \cdot \mathbf{P}_j. \tag{5.7}$$

This can also be presented in a matrix form

$$\mathbf{P} = \mathbf{P}^e + \mathbf{H} \cdot \mathbf{P}, \tag{5.8}$$

where

$$\mathbf{H}_{ij} = \frac{\langle \mathcal{T} b_j, W_i^e \rangle}{\langle b_j, W_j^e \rangle} = \langle \mathcal{T} b_j, \tilde{b}_i \rangle \cdot \frac{\langle b_i, W_i^e \rangle}{\langle b_j, W_j^e \rangle} = \mathbf{R}_{ij} \cdot \frac{\langle b_i, W_i^e \rangle}{\langle b_j, W_j^e \rangle}. \tag{5.9}$$

When finite-element techniques are used together with expansion, finite-element representation can either be used to represent the final result [Kel95], or even be involved in the random walk [PM95].

The latter case may correspond either to the random-walk solution of the linear equation derived by projecting the integral equation, or to the Monte-Carlo evaluation of the multi-dimensional integral containing both the transport and the projection operators. The second case is preferred, because it does not require matrix $\mathbf{R}$ to be explicitly computed and stored.

The main problem of finite-element representations is that they require a lot of basis functions to accurately approximate high-variation, high-dimensional functions. Not surprisingly, finite-element methods have become really popular only for the diffuse case, where the radiance depends on 2 scalars and is relatively smooth. For solving the non-diffuse case, they are good only if the surfaces are not very specular.

5.1 Galerkin's method

Galerkin's method finds an approximation of the solution by making the error orthogonal to the set of basis functions. It means that the projection of error to the original base is zero. Formally, in Galerkin's method the set of basis functions is the same — except for normalization constants — as the set of adjoint basis functions. A particularly important case is the piece-wise constant approximation, when the surfaces and the directions are partitioned into patches $A_1, A_2, \dots, A_n$ and solid angles $\Omega_1, \Omega_2, \dots, \Omega_m$, respectively. The basis functions are then:

$$b_{ij}(\vec{x}, \omega) = \begin{cases} 1 \text{ if } \vec{x} \in A_i \wedge \omega \in \Omega_j, \\ \\ 0 \text{ otherwise.} \end{cases} \tag{5.10}$$

The adjoint basis functions are:

$$\tilde{b}_{ij}(\vec{x}, \omega) = \begin{cases} 1/(|A_i| \cdot |\Omega_j|) \text{ if } \vec{x} \in A_i \wedge \omega \in \Omega_j, \\ \\ 0 \text{ otherwise.} \end{cases} \tag{5.11}$$

The system of linear equations determining the unknown L_{ij} values is

$$\mathbf{L}_{ij} = \mathbf{L}^e_{ij} + \sum_{k=1}^{n} \sum_{l=1}^{m} \mathbf{L}_{kl} \cdot \mathbf{R}_{ijkl}, \tag{5.12}$$

where

$$\mathbf{R}_{ijkl} = \langle \mathcal{T} b_{kl}(\vec{x}, \omega), \tilde{b}_{ij}(\vec{x}, \omega) \rangle = \frac{1}{|A_i| \cdot |\Omega_j|} \cdot \int\limits_{\Omega_j} \int\limits_{A_i} \int\limits_{\Omega} b_{kl}(h(\vec{x}, -\omega'), \omega') \cdot f_r(\omega', \vec{x}, \omega) \cos\theta'_{\vec{x}} \, d\omega' \, d\vec{x} d\omega. \tag{5.13}$$

5.2 Point collocation method

The **point collocation** method finds the unknown coefficients by requiring the finite-element approximation to be exact at the predefined knot points only. Formally, it uses Dirac-delta type adjoint basis functions which emphasize these knot-points:

$$\tilde{b}_{ij}(\vec{x}, \omega) = \delta_{ij}(\vec{x} - \vec{x}_i, \omega - \omega_j). \tag{5.14}$$

The coefficients of the linear equation can be expressed as follows:

$$\mathbf{R}_{ijkl} = \langle \mathcal{T} b_{kl}(\vec{x}, \omega), \tilde{b}_{ij}(\vec{x}, \omega) \rangle = \int\limits_{\Omega} b_{kl}(h(\vec{x}_i, -\omega'), \omega') \cdot f_r(\omega', \vec{x}_i, \omega_j) \cos\theta'_{\vec{x}_i} \, d\omega'. \tag{5.15}$$

5.3 Finite element methods for the diffuse global illumination problem

If the surfaces have only diffuse reflection and emission — which is a general assumption of the **radiosity method** [CG85] — then the rendering (or the potential) equation has a simplified form:

$$L(\vec{x}) = L^e(\vec{x}) + \int\limits_{\Omega_H} L(h(\vec{x}, -\omega')) \cdot f_r(\vec{x}) \cdot \cos\theta'_{\vec{x}} \, d\omega'. \tag{5.16}$$

In this case, the BRDF and the radiance depend on the surface point, but are independent of the direction, which reduces the inherent dimensionality of the problem and simplifies the finite-element representation:

$$L(\vec{x}, \omega) \approx \sum_{j=1}^{n} L_j \cdot b_j(\vec{x}). \tag{5.17}$$

A widely used approach is the application of piece-wise constant basis functions for which $b_j(\vec{x}) = 1$ if $\vec{x}$ is on surface element A_j and 0 otherwise. An appropriate adjoint basis is $\tilde{b}_j(\vec{x}) = 1/A_j$ if $\vec{x}$ is on surface element A_j and 0 otherwise. Using this set of basis functions, the original rendering equation is projected to the following linear equation:

$$\mathbf{L} = \mathbf{L}^e + \mathbf{R} \cdot \mathbf{L} \tag{5.18}$$

where

$$\mathbf{R}_{ij} = \langle \mathcal{T} b_j, \tilde{b}_i \rangle = \frac{1}{A_i} \cdot \int\limits_{A_i} \int\limits_{\Omega} b_j(h(\vec{x}, -\omega')) \cdot f_r(\vec{x}) \cdot \cos\theta'_{\vec{x}} \, d\omega' \, d\vec{x}. \tag{5.19}$$

Let us replace the directional integral by a surface integral using equation (2.2):

$$d\omega' = \frac{d\vec{y} \cdot \cos\theta_{\vec{y}}}{|\vec{x}-\vec{y}|^2}.$$

This is true only when $\vec{x}$ and $\vec{y}$ are visible from each other, otherwise the solid angle of the visible surface is obviously zero. In order to extent the formula to this case, a **visibility indicator** $v(\vec{x},\vec{y})$ is introduced, which is 1 if the two points are visible from each other and zero otherwise. Using this substitution we obtain

$$\mathbf{R}_{ij} = \frac{1}{A_i} \cdot \int_{A_i} \int_{S} b_j(\vec{y}) \cdot f_r(\vec{x}) \cdot \frac{\cos\theta'_{\vec{x}} \cdot \cos\theta_{\vec{y}}}{|\vec{x}-\vec{y}|^2} \cdot v(\vec{x},\vec{y})\, d\vec{y}\, d\vec{x}. \tag{5.20}$$

Taking advantage that the base functions are zero except for their specific domain, $\cos\theta_{\vec{y}} = \cos\theta_j$ and $\cos\theta'_{\vec{x}} = \cos\theta_i$ are constant on these patches, and assuming that the BRDF on patch i is f_i, we get

$$\mathbf{R}_{ij} = \frac{f_i}{A_i} \cdot \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cdot \cos\theta_j}{|\vec{x}-\vec{y}|^2} \cdot v(\vec{x},\vec{y})\, d\vec{y}\, d\vec{x}. \tag{5.21}$$

Note that $\mathbf{R}_{ij}$ is a product of two factors, the albedo of patch i — that is $a_i = f_i \cdot \pi$ —, and a so called **form factor** $\mathbf{F}_{ij}$ which describes the geometry of the scene:

$$\mathbf{F}_{ij} = \frac{1}{A_i} \cdot \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cdot \cos\theta_j}{\pi \cdot |\vec{x}-\vec{y}|^2} \cdot v(\vec{x},\vec{y})\, d\vec{y}\, d\vec{x}. \tag{5.22}$$

So far we have discussed the light propagation problem from the point of view of gathering. For shooting, similar formulae can be produced if incoming direction ω' is replaced by the outgoing direction $\omega = -\omega'$ in equation (5.19):

$$\mathbf{R}_{ij} = \frac{1}{A_i} \cdot \int_{A_i} \int_{\Omega} b_j(h(\vec{x},\omega)) \cdot f_r(\vec{x}) \cdot \cos\theta_{\vec{x}}\, d\omega\, d\vec{x}. \tag{5.23}$$

An adjoint equation can also be derived as a special case of equation (5.8). Let W_i^e be 1 in points of A_i and at the directions of the hemisphere of A_i. The **power equation** is then

$$\mathbf{P} = \mathbf{P}^e + \mathbf{H} \cdot \mathbf{P}, \tag{5.24}$$

where the different forms of $\mathbf{H}_{ij}$ are taken from equation (5.9):

$$\mathbf{H}_{ij} = \mathbf{R}_{ij} \cdot A_i/A_j = \mathbf{R}_{ji} \cdot f_i/f_j = \frac{f_i}{A_j} \cdot \int_{A_j} \int_{\Omega} b_i(h(\vec{y},\omega)) \cdot \cos\theta_j\, d\omega\, d\vec{y}. \tag{5.25}$$

In order to solve the projected integral equation or the power equation, basically the same techniques can be applied as for the original integral equation: inversion, expansion and iteration.

Both the number of unknown variables and the number of equations are equal to the number of surfaces (n). The calculated $\mathbf{L}_i$ radiances represent the light density of the surface on a given wavelength. To generate color pictures at least three independent wavelengths should be selected (say red, green and blue), and the color information will come from the results of the three different calculations.

Thus, to sum up, the basic steps are these:

1. $\mathbf{F}_{ij}$ form factor calculation.
2. Describe the light emission ($\mathbf{L}_i^e$) on the representative wavelengths, or in the simplified case on the wavelength of red, green and blue colors.
3. Solve the linear equation for representative wavelengths.
4. Generate the picture taking into account the camera parameters by any known hidden surface algorithm.

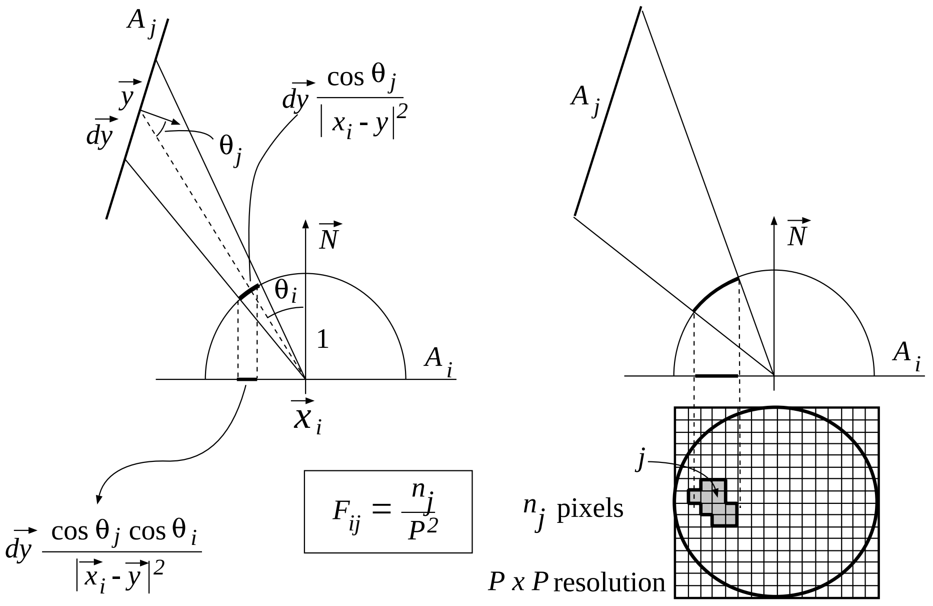
In practical circumstances the number of elemental surface patches can be very large, making the form factor computation and the solution of the linear equation rather time consuming.

5.3.1 Geometric methods for form factor computation

Geometric form factor calculation methods approximate the outer integral of the double integral of the form factor from a single value, thus they apply the following approximation:

$$\mathbf{F}_{ij} = \frac{1}{A_i} \cdot \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cdot \cos\theta_j}{\pi \cdot |\vec{x} - \vec{y}|^2} \cdot v(\vec{x}, \vec{y})\, d\vec{y}\, d\vec{x} \approx \int_{A_j} \frac{\cos\theta_i \cdot \cos\theta_j}{\pi \cdot |\vec{x}_i - \vec{y}|^2} \cdot v(\vec{x}_i, \vec{y})\, d\vec{y}. \tag{5.26}$$

where $\vec{x}_i$ is the center of patch i.



A_j
$\vec{y}$
$\vec{dy}$
θ_j
$\vec{dy}\,\frac{\cos\theta_j}{|x_i - y|^2}$
$\vec{N}$
θ_i
1
A_i
$\vec{x}_i$
$\vec{dy}\,\frac{\cos\theta_j \cos\theta_i}{|\vec{x}_i - \vec{y}|^2}$
$F_{ij} = \frac{n_j}{P^2}$
A_j
$\vec{N}$
A_i
j
n_j pixels
$P \times P$ resolution

Figure 5.3: Geometric interpretation of hemisphere form factor algorithm and the discrete algorithm for form factor computation

Nusselt [SH81] has realized that this formula can be interpreted as projecting the visible parts of A_j onto the unit hemisphere centered above $\vec{x}_i$, then projecting the result orthographically onto the base circle of this hemisphere in the plane of $\vec{x}_i$, and finally calculating the ratio of the doubly projected area and the area of the unit circle (π). Due to the central role of the unit hemisphere, this method is called the **hemisphere algorithm**.

This means that a single row of the form factor matrix can be determined by solving a visibility problem, where the "window" is a half sphere and the "eye" is in the center of surface i.

Continuous hemisphere algorithm

First an exact algorithm is presented for the calculation of the **hemispherical projection** [SKe95].

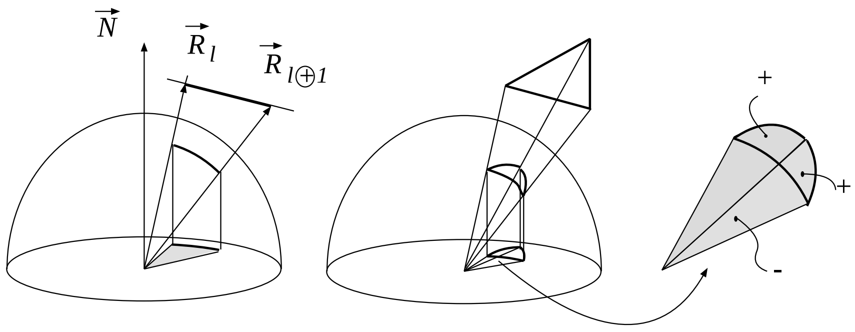


Figure 5.4: Hemispherical projection of a planar polygon ($\vec{R}_l$ and $\vec{R}_{l\oplus 1}$ are two consecutive vertices of the polygon)

To simplify the problem, consider only one edge line of the polygon first, and two consecutive vertices, $\vec{R}_l$ and $\vec{R}_{l\oplus 1}$, on it (figure 5.4). Operator $\oplus$ stands for modulo L addition which can handle the problem that the next of vertex l is usually vertex $l+1$, except for vertex $L-1$ which is followed by vertex 0. The hemispherical projection of this line is a half great circle. Since the radius of this great circle is 1, the area of the sector formed by the projections of $\vec{R}_l$ and $\vec{R}_{l\oplus 1}$ and the center of the hemisphere is simply half the angle of $\vec{R}_l$ and $\vec{R}_{l\oplus 1}$. Projecting this sector orthographically onto the equatorial plane, an ellipse sector is generated, having the area of the great circle sector multiplied by the cosine of the angle of the surface normal $\vec{N}$ and the normal of the segment $(\vec{R}_l \times \vec{R}_{l\oplus 1})$.

The area of the doubly projected polygon can be obtained by adding and subtracting the areas of the ellipse sectors of the different edges, as is demonstrated in figure 5.4, depending on whether the projections of vectors $\vec{R}_l$ and $\vec{R}_{l\oplus 1}$ follow each other clockwise. This sign value can also be represented by a signed angle of the two vectors, expressing the area of the double projected polygon as a summation:

$$\sum_{l=0}^{L-1} \frac{1}{2} \cdot \mathrm{angle}(\vec{R}_l, \vec{R}_{l\oplus 1}) \frac{(\vec{R}_l \times \vec{R}_{l\oplus 1})}{|\vec{R}_l \times \vec{R}_{l\oplus 1}|} \cdot \vec{N}. \tag{5.27}$$

Having divided this by π to calculate the ratio of the area of the double projected polygon and the area of the equatorial circle, the form factor can be generated.

This method has supposed that surface A_j is above the plane of A_i and is totally visible. Surfaces below the equatorial plane do not pose any problems, since we can get rid of them by the application of a clipping algorithm. When partial occlusion occurs, then either a continuous (object precision) visibility algorithm is used to select the visible parts of the surfaces, or the visibility term is estimated by firing several rays to surface element j and averaging their 0/1 associated visibilities [Tam92].

Discrete hemisphere algorithm and its variations

Discrete methods subdivide the hemisphere into finite number of areas called “pixels” and assume that what can be seen through these small areas is homogeneous (figure 5.3). Thus it is enough to determine the visibility through a single point in each pixel.

The complicated form of the “hemispherical window” can be simplified if the hemisphere is replaced by other immediate surfaces, such as a **hemicube** [CG85] or a **cubic tetrahedron** [BKP91]. In these cases the modification of the geometry must be compensated by appropriate weighting functions in area computation. For hemicube and hemishpere algorithms, the window surface consists of normal rectangles, which can be exploited by the built in transformation and scan-conversion hardware of graphics workstations.

Figure 5.5: A complex scene rendered by the hemicube algorithm (University of Cornell)

Chapter 6

Numerical quadrature for high dimensional integrals

The solution of the rendering equation requires the numerical evaluation of high-dimensional integrals. Numerical quadrature formulae take finite samples from the domain, evaluate the integrand for these samples and generate the integral as a weighted sum of the integrand values. The general form for the solution is

$$I = \int_V f(\mathbf{z})\, d\mathbf{z} \approx \sum_{i=1}^{N} f(\mathbf{z}_i) \cdot w(\mathbf{z}_i) \tag{6.1}$$

where $\mathbf{z}_i$ is a sample point from the s-dimensional domain V, and w is the weighting function of the given quadrature rule.

Classical quadrature rules usually trace back the computation of multi-dimensional integrals to a series of one-dimensional integrals. Thus they select the coordinates of the sample points independently, and the high-dimensional points are defined as the Cartesian product of the 1-dimensional sample sets. The simplest techniques, including the **brick-rule**, the **trapezoidal-rule**, **Simpson-rule**, etc. space the abscissas equally, which results in a regular grid in the domain. More sophisticated techniques, such as the **Gaussian quadrature**, selects the sample points non uniforly along each abscissa to decrease the error of the integral.

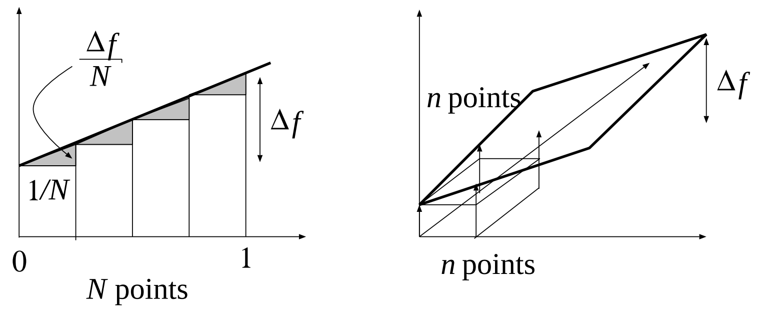


Figure 6.1: Error of the brick rule in one and two dimensional spaces

Let us evaluate the error of the brick rule in one and two dimensional spaces (figures 6.1). Suppose that N sample points are used in the domains of $[0, 1]$ and $[0, 1]^2$, respectively. In the one dimensional space the error is the sum of the areas of triangles that are between the function and the series of bricks. The average height of a triangle is $\Delta f/(2N)$ where Δf is the variation of the integrand and N is the number of bricks. Since the base of a triangle is $1/N$ and the number of triangles is N, the error is:

$$\epsilon_1 = \frac{\Delta f}{2N} \cdot \frac{1}{N} \cdot N = \frac{\Delta f}{2N} = \mathcal{O}\left(\frac{1}{N}\right). \tag{6.2}$$

In the two dimensional space the N sample points should be distributed along two coordinates, thus we have just $n = \sqrt{N}$ points in a single row and column. Now the error is the sum of the volume of the roof-like objects that are between the function and the series of bricks. The average height of a volume element is $\Delta f/(2n)$, its base has $1/n^2$ area and there are n^2 elements. Thus the error is

$$\epsilon_2 = \frac{\Delta f}{2n} \cdot \frac{1}{n^2} \cdot n^2 = \frac{\Delta f}{2\sqrt{N}} = \mathcal{O}\left(\frac{1}{\sqrt{N}}\right). \tag{6.3}$$

This idea can simply generalized to any dimensions and we can conclude that the integration error of the brick rule in an s-dimensional space is proportional to $\Delta f/N^{1/s}$ (iΔf is the total change, i.e. the variation, of the integrand). From another point of view, it means that the required number of sample points to find an estimate with error ϵ is proportional to $(\Delta f/\epsilon)^s$, thus the computational complexity is exponential with regard to the dimension of the domain. This phenomenon is called as the **dimensional explosion** or **dimensional core**. Concerning other classical quadrature rules, Δf measures higher order changes, such as the distance from the piece-wise linear or polynomial functions, but they also exhibit dimensional core.

The dimensional explosion can be avoided by **Monte-Carlo** [Sob91] or **quasi-Monte Carlo** [Nie92] integration methods. Monte-Carlo methods trace back the estimation of an integral to the calculation of an expected value, which is estimated by averaging random samples. Quasi-Monte Carlo techniques, on the other hand, use deterministic samples that are uniformly distributed in the integration domain.

6.1 Monte-Carlo quadrature

Monte-Carlo integration converts the calculation of an integral to an equivalent expected value problem. Assume that a random vector variable $\mathbf{z}$ is uniformly distributed in V, thus its probability density is $p(\mathbf{z}) = 1/V$. The expected value of random variable $f(\mathbf{z})$ is

$$E[f(\mathbf{z})] = \int\limits_V f(\mathbf{z}) \cdot p(\mathbf{z})\, d\mathbf{z} = \int\limits_V f(\mathbf{z}) \cdot \frac{1}{V}\, d\mathbf{z} = \frac{1}{V} \cdot I, \tag{6.4}$$

thus the required integral can easily be found if this expected value is available. According to the **theorems of large numbers**, if independent random samples $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N$ are generated using probability density p, then the expected value can be estimated by the average $\hat{f}$:

$$E[f(\mathbf{z})] \approx \hat{f} = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{z}_i). \tag{6.5}$$

Estimator $\hat{f}$ is also a random variable whose expected value is equal to $E[f(\mathbf{z})] = I/V$. Suppose that the variance of $f(\mathbf{z})$ is σ^2. If the samples are independent random variables, then the variance of estimator $\hat{f}$ is:

$$D^2\left[\hat{f}\right] = \frac{1}{N^2} \sum_{i=1}^{N} \sigma^2 = \frac{\sigma^2}{N}. \tag{6.6}$$

Thus the standard deviation of the estimator is

$$D\left[\hat{f}\right] = \frac{\sigma}{\sqrt{N}}. \tag{6.7}$$

According to the **central limit theorem**, estimator $\hat{f}$ will have normal distribution asymptotically, with mean $E[f(\mathbf{z})] = I/V$ and standard deviation $\sigma/\sqrt{N}$. Examining the shape of the probability density of the normal distribution, we can conclude that the probability that the distance between the variable and the mean is less than 3 times the standard deviation is 0.997. Thus with 0.997 confidence level we can say that the (probabilistic) error bound of the Monte-Carlo quadrature is

$$|\int\limits_V f(\mathbf{z})\, d\mathbf{z} - \frac{V}{N} \sum_{i=1}^{N} f(\mathbf{z}_i)| < \frac{3V\sigma}{\sqrt{N}}. \tag{6.8}$$

Let us realize that this bound is independent of the dimension of the domain! This property means that by Monte-Carlo quadrature the dimensional explosion can be avoided.

6.2 Quasi-Monte Carlo quadrature

In the previous section we concluded that randomly distributing the sample points solves the dimensional core problem. We show that it is also possible with deterministically selected sample points. The resulting method is called **quasi-Monte Carlo quadrature**. For the sake of simplicity, assume that an s-dimensional funcion $f(\mathbf{z})$ needs to be integrated over the domain $[0,1]^s$. In order to guarantee that f is Riemann-integrable, f is assumed to be piece-wise continuous.

This integral is approximated by a finite sum as follows:

$$\int\limits_{\mathbf{z}\in[0,1]^s} f(\mathbf{z})\, d\mathbf{z} \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i). \tag{6.9}$$

The question is how the sequence of sample points $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N$ should be selected to minimize the error of the integral quadrature. A minimal requirement is the stability of the sequence, which means that in asymptotic sense the error is zero for any Riemann integrable function:

$$\int\limits_{\mathbf{z}\in[0,1]^s} f(\mathbf{z}) d\mathbf{z} = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i). \tag{6.10}$$

Sequences meeting this requirement are called **uniform** or **equidistribution** (or more precisely **1-equidistribution**) sequences [Nie92].

In order to find other necessary requirements for uniform sequences, let us consider the integration of a very simple function which is 1 inside a d-dimensional "brick" originating at the center of the coordinate system and 0 elsewhere:

$$\mathcal{L}(\mathbf{z}) = \begin{cases} 1 \text{ if } 0 \leq \mathbf{z}|_1 \leq v_1, 0 \leq \mathbf{z}|_2 \leq v_2, \ldots, 0 \leq \mathbf{z}|_s \leq v_s, \\ \\ 0 \text{ otherwise.} \end{cases} \tag{6.11}$$

Let us denote the volume of this brick by $V(A) = \prod_{j=1}^{s} v_j$. Integrating this function we have:

$$\int\limits_{\mathbf{z}\in[0,1]^s} \mathcal{L}\, d\mathbf{z} = \prod_{j=1}^{s} v_j = V(A). \tag{6.12}$$

If the number of sample points that are inside the s-dimensional "brick" A is $m(A)$, then the finite approximation sum is

$$\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i) = \frac{m(A)}{N}. \tag{6.13}$$

Since the exact value of the integral is $V(A)$ now, for uniform sequences, the average number of sample points that are located inside a volume should be proportional to the size of this volume:

$$\lim_{N\to\infty} \frac{m(A)}{N} = V(A). \tag{6.14}$$

If the size of the sequence of the sample points is not infinite, then the proportionality requirement can only be approximately met. The maximal error in this approximation is called the **discrepancy** (or the **star-discrepancy**) of the sequence:

$$\mathcal{D}^*(\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_N) = \sup_A |\frac{m(A)}{N} - V(A)|. \tag{6.15}$$

If a sequence is appropriate for integral-quadrature, then the approximation sum is expected to converge to the real value of the integral, which requires the discrepancy to converge to 0. This is another necessary requirement which is derived from the examination of a very simple function. Note that this requirement also emphasizes the uniformity of those sequences that can be used for numerical integration.

However, this requirement is not only necessary, but also sufficient, since any Riemann-integrable function can be approximated by piece-wise constant step-functions with arbitrary precision. To show how step-functions can do this, an example of a 2-dimensional function is shown in figure 6.2.

6.2.1 Error Analysis for integrands of finite variation: Koksma-Hlawka Inequality

In this section, an upper-bound is given to the error of the quasi-Monte Carlo quadrature for functions that have bounded and piece-wise continuous mixed derivatives.

The error of the quadrature is shown below:

$$|\int\limits_{\mathbf{z}\in[0,1]^s} f(\mathbf{z})\, d\mathbf{z} - \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i)|. \tag{6.16}$$

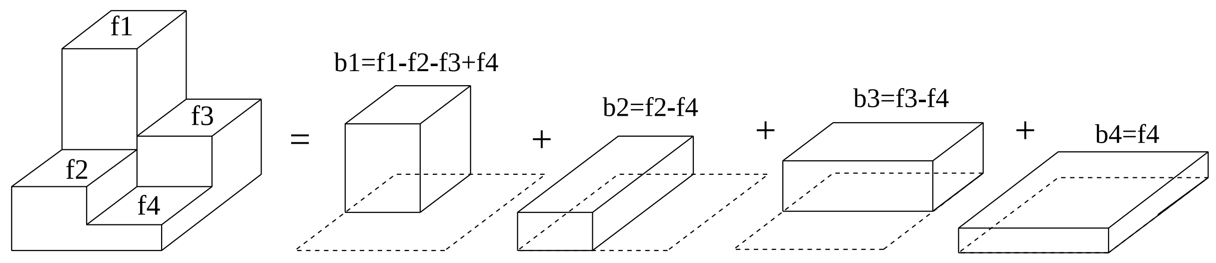


Figure 6.2: Definition of functions from bricks originating at the center of the coordinate system

Intuitively this error must depend on two independent factors. On the one hand, if the discrepancy of the sequence of sample points is high, then there are large regions where there are no sample point at all, which increases the error. This means that the error is expected to be proportional to the discrepancy of the sequence of sample locations.

On the other hand, the error also depends on how quickly the function changes between the sample points. If the function can change significantly in small domain, then the error can be quite large. However, if the slope of the function is small, then nothing dramatic happens between the sample points thus the error will be small.

Measures describing how rapidly a function can change are called **variation**s. For a 1-dimensional function the **variation in the sense of Vitali** is defined as:

$$\mathcal{V}_{\mathrm{V}}(f(x)) = \limsup \sum_{i=1}^{N} |f(x_{i+1}) - f(x_i)|. \tag{6.17}$$

For a 2-dimensional function, the definition is analogous:

$$\mathcal{V}_{\mathrm{V}}(f(x,y)) = \limsup \sum_{i=1}^{N} \sum_{j=1}^{M} |f(x_{i+1}, y_{j+1}) - f(x_{i+1}, y_i) - f(x_i, y_{i+1}) + f(x_i, y_i)|. \tag{6.18}$$

Note that for higher dimensions, the variation of Vitali does not provide a good measure: if the function is constant in x, for instance, then the variation is zero, regardless how the function changes depending on y.

Thus, it is worth introducing a somehow more stronger variation type, called the **Hardy-Krause variation**. The variation in the sense of Hardy-Krause is the sum of the variations of the function and of its restrictions to the end of the domain. For dimension 2, the new variation is:

$$\mathcal{V}_{\mathrm{HK}}(f(x,y)) = \mathcal{V}_{\mathrm{V}} f(x,y) + \mathcal{V}_{\mathrm{V}} f(x,1) + \mathcal{V}_{\mathrm{V}} f(1,y). \tag{6.19}$$

If a function has bounded and piece-wise continuous mixed derivatives, then its variation is finite. For a 2-dimensional function meeting this requirement the variation can be given by the following formula:

$$\mathcal{V}_{\mathrm{HK}}(f(u,v)) = \int_0^1 \int_0^1 \left| \frac{\partial^2 f(u,v)}{\partial u \partial v} \right| \, du \, dv + \int_0^1 \left| \frac{\partial f(u,1)}{\partial u} \right| \, du + \int_0^1 \left| \frac{\partial f(1,v)}{\partial v} \right| \, dv. \tag{6.20}$$

The property that a function is not continuous does not necessarily mean that the variation is infinite. If at most finite or countable infinite discontinuities occur at hyper-planes parallel to the coordinate axes, then the variation is still finite. An example of a discontinuous function that have finite variation is

$$f(x,y) = \begin{cases} 1 \text{ if } x > x_0, \\ 0 \text{ otherwise.} \end{cases} \tag{6.21}$$

However, when the discontinuity is not parallel to the coordinate axes, then the variation is infinite. A simple function of infinite variation is [Deá89]:

$$f(x,y) = \begin{cases} 1 \text{ if } x > y, \\ 0 \text{ otherwise.} \end{cases} \tag{6.22}$$

Now, let us turn to the error of quasi-Monte Carlo integration. The following formula, which expresses the previous intuition that the error depends on the uniformness of the sample points and on the variation of the integrand, is called the **Koksma-Hlawka inequality**:

$$|\int\limits_{\mathbf{z}\in[0,1]^s} f(\mathbf{z})\,d\mathbf{z} - \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i)| \leq \mathcal{V}_{\mathrm{HK}} \cdot \mathcal{D}^*(\mathbf{z}_1, \mathbf{z}_2, \dots \mathbf{z}_N). \tag{6.23}$$

For the sake of notational simplicity, the Koksma-Hlawka inequality is proven here only for the two-dimensional case ($s = 2$). The outline of the proof is as follows. We start with the assumption that the function has piece-wise continuous and bounded mixed derivatives, thus the mixed derivatives are Riemann-integrable and their integration gives back the original function. First the function is expressed as the integral of its derivatives, and the function values in the approximation sum are converted accordingly. On the other hand, the integral of f is converted to a similar form using partial integration. Finally the difference of the approximating sum and the integral is examined and upperbounded.

First, the value of function $f(u, v)$ is expressed by its derivatives at point x, y:

$$f(x,y) = f(1,1) + [f(1,1) - f(x,1) - f(1,y) + f(x,y)] - [f(1,1) - f(x,1)] - [f(1,1) - f(1,y)] =$$

$$f(1,1) + \int\limits_x^1\int\limits_y^1 f_{uv}(u,v)\,du\,dv - \int\limits_x^1 f_u(u,1)\,du - \int\limits_y^1 f_v(1,v)\,dv.$$

where

$$f_{uv}(u,v) = \frac{\partial^2 f(u,v)}{\partial u \partial v}, \quad f_u(u,v) = \frac{\partial f(u,v)}{\partial u}, \quad f_v(u,v) = \frac{\partial f(u,v)}{\partial v}.$$

Let us introduce the step function ϵ:

$$\epsilon(u,v) = \begin{cases} 1 \text{ if } u \geq 0, v \geq 0, \\ 0 \text{ otherwise.} \end{cases}$$

Using this step function, the domains of the integrals can be extended to [0, 1], as follows:

$$f(x,y) = f(1,1) + \int\limits_0^1\int\limits_0^1 f_{uv}(u,v)\cdot\epsilon(u-x,v-y)\,du\,dv - \int\limits_0^1 f_u(u,1)\cdot\epsilon(u-x,1)\,du - \int\limits_0^1 f_v(1,v)\cdot\epsilon(1,v-y)\,dv.$$

Substituting $\mathbf{z}_i = (x_i, y_i)$ into this formula we have:

$$f(\mathbf{z}_i) = f(1,1) + \int\limits_0^1\int\limits_0^1 f_{uv}(u,v)\cdot\epsilon(u-x_i,v-y_i)\,du\,dv - \int\limits_0^1 f_u(u,1)\cdot\epsilon(x_i,1)\,du - \int\limits_0^1 f_v(1,v)\cdot\epsilon(1,y_i)\,dv.$$

Averaging these formulae for $i = 1, 2, \dots N$, the approximating sum has the following form:

$$\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i) = f(1,1) + \frac{1}{N}\int\limits_0^1\int\limits_0^1 f_{uv}(u,v)\cdot m(u,v)\,du\,dv - \frac{1}{N}\int\limits_0^1 f_u(u,1)\cdot m(u,1)\,du - \frac{1}{N}\int\limits_0^1 f_v(1,v)\cdot m(1,v)\,dv.$$

where

$$m(u,v) = \sum_{i=1}^{N} \epsilon(u-x_i, v-y_i),$$

which is the number of points located in the rectangle $[(0,0),(u,v)]$. The integral

$$\int\limits_{\mathbf{z}\in[0,1]^2} f(\mathbf{z})\,d\mathbf{z} = \int\limits_{v=0}^1\int\limits_{u=0}^1 f(u,v)\,du\,dv$$

can also be converted to a similar form, if partial integration is applied. First the inner integral is considered:

$$\int\limits_{u=0}^1 f(u,v)\,du = \int\limits_{u=0}^1 f(u,v)\cdot 1\,du = f(1,v) - \int\limits_{u=0}^1 f_u(u,v)\cdot u\,du = F(v)$$

Then the outer integral is processed in a similar way:

$$\int\limits_{v=0}^1\int\limits_{u=0}^1 f(u,v)\,du\,dv = \int\limits_{v=0}^1 F(v)\,dv = \int\limits_{v=0}^1 F(v)\cdot 1\,dv = F(1) - \int\limits_{v=0}^1 F_v(v)\cdot v\,dv$$

Substituting the definition of $F(v)$ we get:

$$\int_{v=0}^{1}\int_{u=0}^{1} f(u,v)\,du\,dv = f(1,1) + \int_0^1\int_0^1 f_{uv}(u,v)\cdot uv\,du\,dv - \int_0^1 f_u(u,1)\cdot u\,du - \frac{1}{N}\int_0^1 f_v(1,v)\cdot v\,dv.$$

Thus the error of quadrature is then:

$$|\int_{v=0}^{1}\int_{u=0}^{1} f(u,v)\,du\,dv - \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i)| =$$

$$|\int_0^1\int_0^1 f_{uv}(u,v)\cdot\left(\frac{m(u,v)}{N} - uv\right)\,du\,dv - \int_0^1 f_u(u,1)\cdot\left(\frac{m(u,1)}{N} - u\right)\,du - \int_0^1 f_v(1,v)\cdot\left(\frac{m(1,v)}{N} - v\right)\,dv| \le$$

$$\left(\int_0^1\int_0^1 |f_{uv}(u,v)|\cdot\,du\,dv + \int_0^1 |f_u(u,1)|\,du + \int_0^1 |f_v(1,v)|\,dv\right)\cdot\sup_{u,v}\left|\left(\frac{m(u,v)}{N} - uv\right)\right| = \mathcal{V}_{\mathrm{HK}}\cdot\mathcal{D}^*(\mathbf{z}_1,\mathbf{z}_2,\ldots\mathbf{z}_N).$$

This is exactly what we wanted to prove.

According to this inequality, the error can be upperbounded by the product of two independent factors, the variation of the integrand and the discrepancy of the used sample set. The discrepancy shows how uniformly the set is distributed [Shi91a]. This immediately presents two orthogonal strategies to improve the quality of quadratures. Either we try to make the function flat by appropriate variable transformations, or use very uniformly distributed sample sets. The first technique is called **importance sampling** [Sob91], while the second involves the **stratification** [Sob91, Mit96, Arv95] of random points or the application of **low-discrepancy series** [Nie92, War95, PFTV92, Knu81, Sob91].

If the integrand has finite variation, then the error is proportional to the discrepancy of the sequence of sample locations. For carefully selected sample points the discrepancy can converge to zero with almost linear speed. Thus, quadratures having almost linear convergence can be obtained in this way, which is better than the $\mathcal{O}(1/\sqrt{N})$ speed of Monte-Carlo quadratures.

Note, on the other hand, that functions having infinite variation can also be integrated by quasi-Monte Carlo quadrature. The quadrature will be asymptotically exact for any uniform sequence and for any Riemann integrable function. The fact that the Koksma-Hlawka inequality cannot be applied means that it cannot be used to provide a qualitative measure for the speed of the convergence. Practical experience shows that quasi-Monte Carlo integration outperforms the classical Monte-Carlo integration even for discontinuous functions [SKDP99]. However, the difference in the effectiveness becomes significantly less when the integrand has infinite variation. This phenomenon will be investigated in the subsequent sections.

Since the integrand of the rendering and potential equations is usually discontinuous, this case is very important for computer graphics applications.

6.2.2 Generation of the sample points

As a conclusion of error analysis we can state that we need very uniform sequences for quasi-Monte Carlo quadrature. Regular grids should be rejected because of their $\mathcal{O}(1/N^{1/s})$ discrepancy which results in dimensional explosion.

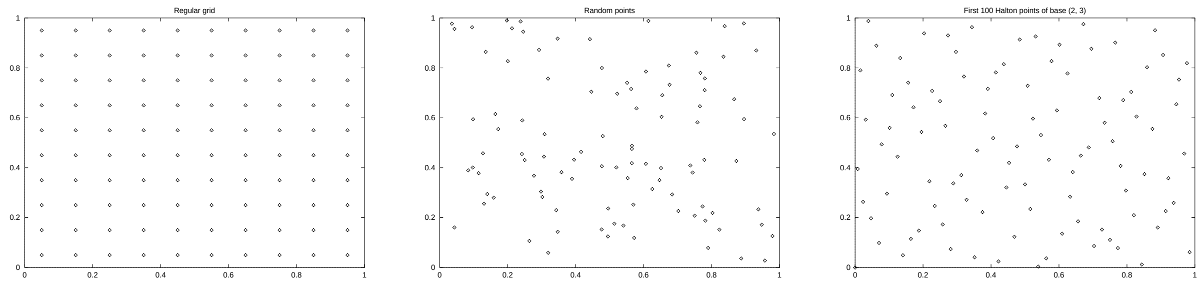


Figure 6.3: 100 points distributed by a regular grid (left), random distribution (middle) and Halton low-discrepancy sequence (right)

Monte-Carlo method proposed the application of random samples to avoid dimensional explosion. This can also be justified by analyzing the discrepancy. In fact, the discrepancy of a uniformly distributed random series is

$$\mathcal{O}\left(\sqrt{\frac{\log\log N}{2N}}\right)$$

assymptotically with probability 1.

In order to prove this, let us define a new random variable ξ_i from uniformly distributed random sample $\mathbf{z}_i$ in the following way:

$$\xi_i = \begin{cases} 1 \text{ if } \mathbf{z}_i \text{ is in an } s\text{-dimensional brick } A \text{ that originates at the center,} \\ \\ 0 \text{ otherwise.} \end{cases}$$

Note that if $\mathbf{z}_i$ is uniformly distributed in [0, 1], then the expected value and the variance of ξ_i are

$$E[\xi_i] = V(A), \quad D^2[\xi_i] = V(A) - V^2(A) \le \frac{1}{4},$$

where $V(A)$ is the volume of brick A. If the samples are independent random variables, the generated $\xi_1, \xi_2, \ldots, \xi_N$ random variables will also be independent, and of the same distribution having mean $E[\xi]$ and variance $D^2[\xi]$.

According to the **theorem of iterated logarithm** [Rén81], the difference between the average of independent random variables $\xi_1, \xi_2, \ldots, \xi_N$ of the same distribution and their mean $E[\xi]$ can be upperbounded in the following way:

$$\Pr\left\{\limsup \left|\sum \frac{\xi_i}{N} - E[\xi]\right| \le \sqrt{D^2[\xi] \cdot \frac{2\log\log N}{N}}\right\} = 1,$$

In our case the standard deviation $D^2[\xi]$ cannot exceed $1/4$ and

$$\sup \left|\sum \frac{\xi_i}{N} - E[\xi]\right| = \sup_A \left|\frac{m(A)}{N} - V(A)\right| = \mathcal{D}^*(\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_N),$$

thus the theorem of iterated logarithm becomes what we want to prove:

$$\Pr\left\{\lim \mathcal{D}^*(\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_N) \le \sqrt{\frac{\log\log N}{2N}}\right\} = 1.$$

6.2.3 Generation of low-discrepancy sequences

Beyond random sequences, however, there are deterministic sequences that have even better discrepancy. The discrepancy of the best sequences known is in the order of $\mathcal{O}(\log^s N/N)$ or even in the order of $\mathcal{O}(\log^{s-1} N/N)$ if N is known before starting the sequence. These sequences are called **low-discrepancy** sequences. There are many sequences published in the literature [Nie92, War95, Deá89, Knu81]. The most famous one is probably the **Halton-sequence** (its one-dimensional version is also called **Van der Corput sequence**).

The element i of the one-dimensional Halton sequence of base b is defined as the radical inverse of the expansion of i in base b. This means that number i is expanded in radix b, then the number is mirrored onto the "radix" point. The first few points in base 2 are shown in table 6.1.

i	binary form of i	radical inverse	H_i
1	1	0.1	0.5
2	10	0.01	0.25
3	11	0.11	0.75
4	100	0.001	0.125
5	101	0.101	0.625
6	110	0.011	0.375
7	111	0.111	0.875

Table 6.1: The calculation of the ith Halton point H_i in base 2

Why is this sequence uniform? Note that the construction algorithm generates as binary form of i all binary combinations of length k before producing a combination of length $k+1$. This means that after the radical inverse the sequence H_i will visit all intervals of length 2^{-k} before putting a new point in an interval already visited. Thus the sequence is really uniform.

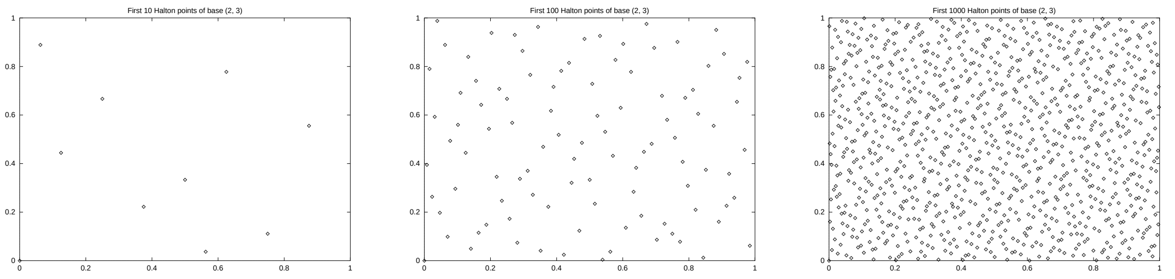


Figure 6.4: The distribution of the first 10,100 and 1000 Halton points in 2 dimensions

On the other hand, as k increases, the algorithm produces a single point in each interval of length $1/2$, then in each interval of length $1/4$, etc. thus the sequence is not only asymptotically uniform, but also the first N points are fairly uniformly distributed (this is guaranteed by the property that the radical inverse makes the most significant bit the most rapidly changing bit). This is also true in other radix systems as well. If the base is b, then the Halton sequence will place a sample point in all intervals of size b^{-k} before putting a new point into any intervals.

A Halton sequence is able to generate points that are uniformly distributed in the 1-dimensional $[0, 1]$ interval. If higher dimensional regions, such as rectangles, cubes, etc. should be filled uniformly, then different coordinates of the sample vectors can be generated from Halton sequences of different base numbers. In order for these vectors to uniformly fill the whole region, the "interdependence" of different coordinates should be as little as possible. To examine this, let us assume that a two-dimensional Halton sequence is generated with base numbers b_1 and b_2. According to the behavior of the Halton sequence, the generation algorithm would visit all columns of width $b_1^{-k_1}$ before visiting a column again, and similarly it would visit all rows of height $b_2^{-k_2}$ before putting a new point into a row. The columns and rows form $b_1^{k_1} \cdot b_2^{k_2}$ cells. Uniformness in the two-dimensional space means that the algorithm is expected to put a point in each cell before putting a second point in any cells. Since the periodicity of the columns and rows are $b_1^{k_1}$ and $b_2^{k_2}$, respectively, the periodicity of the cells is the smallest common multiple of $b_1^{k_1}$ and $b_2^{k_2}$. This equals to the their product, that is total number of cells, if b_1 and b_2 are relative primes.

This can also be stated in a general way. If a multi-dimensional Halton sequence is to be constructed, then the base numbers of different coordinates must be relative primes.

A C++ class that can initialize an arbitrary Halton point and then it can generate incrementally all subsequent points using a very fast algorithm [Kel96b] is presented in the following:

```
class Halton {
        double value, inv_base;

        Number( long i, int base ) {
                double f = inv_base = 1.0/base;
                value = 0.0;
                while ( i > 0 ) {
                        value += f * (double)(i % base);
                        i /= base;
                        f *= inv_base;
                }
        }

        void Next( ) {
                 double r = 1.0 - value - 0.0000000001;
                 if (inv_base < r) value += inv_base;
                 else {
                        double h = inv_base, hh;
                        do {
                                hh = h;
                                h *= inv_base;
                        } while ( h >= r );
                        value += hh + h - 1.0;
                }
        }
        operator double() { return value; }
};
```

6.3 Importance sampling

Importance sampling is a well-known method of Monte-Carlo integration to reduce variance. The basic idea is to use non-uniform distribution to find sample points, which places more samples where the function is large. More specifically it means that

$$I = \int_V f(\mathbf{z})\, d\mathbf{z} = \int_V \frac{f(\mathbf{z})}{p(\mathbf{z})} \cdot p(\mathbf{z})\, d\mathbf{z} = E\left[\frac{f(\mathbf{z})}{p(\mathbf{z})}\right] \approx \frac{1}{N} \cdot \sum_{i=1}^{N} \frac{f(\mathbf{z}_i)}{p(\mathbf{z}_i)} \pm \frac{3V\sigma}{\sqrt{N}}, \tag{6.24}$$

where $p(\mathbf{z})$ is a probability density in V, the $\mathbf{z}_i$ points are selected according to this probability density, and the variance σ^2 is defined by

$$\sigma^2 = D^2\left[\frac{f(\mathbf{z})}{p(\mathbf{z})}\right] = E\left[\left(\frac{f(\mathbf{z})}{p(\mathbf{z})} - I\right)^2\right] = \int_V \left(\frac{f(\mathbf{z})}{p(\mathbf{z})} - I\right)^2 \cdot p(\mathbf{z})\, d\mathbf{z}. \tag{6.25}$$

The probability density $p(\mathbf{z})$ should be selected to minimize the variance. As can be shown easily, the variance can be minimized if $p(\mathbf{z})$ is proportional to the integrand $f(\mathbf{z})$. In order to demonstrate this, let us express the ratio of the integrand and the probability density in the following way:

$$\frac{f(\mathbf{z})}{p(\mathbf{z})} = \alpha + \beta \cdot \delta(\mathbf{z}), \tag{6.26}$$

where $\alpha = E[\frac{f(\mathbf{z})}{p(\mathbf{z})}]$ and $\int_V (\delta(\mathbf{z}))^2 \cdot p(\mathbf{z})\, d\mathbf{z} = 1$. Using this, the variance of the integral quadrature is

$$\sigma^2 = E[(\alpha + \beta \cdot \delta(\mathbf{z}) - E[\alpha + \beta \cdot \delta(\mathbf{z})])^2] = \beta^2 \cdot E[(\delta(\mathbf{z}))^2] = \beta^2. \tag{6.27}$$

This is obviously minimal if $\beta = 0$, when the variance is also zero.

Thus in Monte-Carlo integration it is worth applying probability distributions that are large where the integrand is large and small where the integrand is negligible.

6.3.1 Generation of a random variable with a prescribed probability density

We concluded that importance sampling requires random samples generated from a probability density which is proportional — at least approximately — to the integrand. This section examines how such samples can be found. First, let us consider the 1-dimensional case and assume that the domain of the integration is an interval $[a, b]$.

Suppose that we want samples with a probability density that is proportional to a function $g(z)$. This function is an approximation of the integrand f. If this function is different from the integrand, then the importance sampling is not optimal. The needed probability density $p(z)$ can be obtained by scaling function g to integrate to 1 as probability densities do:

$$p(z) = \frac{g(z)}{\int_a^b g(z)dz}. \tag{6.28}$$

Note that this gives us a clear explanation why we should use non-optimal densities. If g were equal to f, then the construction of the probability density would require the integral of f.

From the probability density p, probability distribution function P is obtained:

$$P(z) = \int_a^z g(Z)\, dZ. \tag{6.29}$$

A random variable ξ having probability distribution P can be constructed by transforming another random variable r, which is uniformly distributed in the $[0, 1]$ interval, with the $\xi = P^{-1}(r)$ transformation.

To prove this, the probability distribution of ξ is examined:

$$\Pr\{\xi < z\} = \Pr\{P^{-1}(r) < z\} = \Pr\{r < P(z)\} = P(z). \tag{6.30}$$

since $P(z)$ is not decreasing and the probability that r is in an interval of $[0, 1]$ equals to the size of this interval.

The multi-dimensional case can be traced back to a series of 1-dimensional constructions. After normalization we have

$$p(\mathbf{z}) = \frac{g(\mathbf{z})}{\int_V g(\mathbf{z})d\mathbf{z}}. \tag{6.31}$$

The probability density is expressed as a product of 1-dimensional conditional densities:

$$p(z_1, z_2, \ldots, z_s) = p_1(z_1|z_2, \ldots, z_s) \cdot p_2(z_2|z_3, \ldots, z_s) \cdot \ldots \cdot p_s(z_s). \tag{6.32}$$

For these conditional densities the same method can be used recursively as for the 1-dimensional case. If the different coordinates are independent random variables, then we obtain:

$$p(z_1, z_2, \ldots, z_s) = p_1(z_1) \cdot p_2(z_2) \cdot \ldots \cdot p_s(z_s). \tag{6.33}$$

6.3.2 Importance sampling in quasi-Monte Carlo integration

Classical Monte-Carlo integration places more samples where the integrand is large. The same basic idea also makes sense in quasi-Monte Carlo integration. However, for formal analysis, we have to find another approach since terms like probability density or variance cannot be applied in the deterministic quasi-Monte Carlo framework.

The alternative formulation is the integration using variable transformation. Suppose that a function f needs to be integrated in domain V and we have a monotonously increasing mapping T that maps this domain onto V'. The integral can also be evaluated in the new domain using the following formula:

$$\int_V f(\mathbf{z})\, d\mathbf{z} = \int_{V'} f(T^{-1}(\mathbf{y})) \left| \frac{\partial T^{-1}(\mathbf{y})}{\partial \mathbf{y}} \right| \, d\mathbf{y}, \tag{6.34}$$

where $\left| \frac{\partial T^{-1}(\mathbf{y})}{\partial \mathbf{y}} \right|$ is the Jacobi determinant of the inverse transformation.

If quasi-Monte Carlo integration is used, then domain V' is $[0,1]^s$. In order to quantify the error of the quadrature, we can use the Koksma-Hlawka inequality which expresses the error-bound as a product of the discrepancy of the sample points and the variation of the function. Since the discrepancy of the sample points is independent of the function to be integrated, the error-bound can be controlled by the appropriate transformation of the integrand to reduce variation. In order to reduce the variation, the function should be flattened. In the ideal case when the integrand is constant, the variation is 0.

To make the transformed integrand constant, the Jacobi determinant should be inversely proportional to f. Since the intuitive meaning of the Jacobi determinant is the compression or expansion ratio of the two corresponding sub-cubes in V and V' respectively, this criterion states that if the sample points are uniformly distributed in V', then the transformation will concentrate them around regions where f is high.

For the sake of simplicity, the details are discussed only for the 1-dimensional case, when the variable transformation has the following form

$$\int_{z_{\min}}^{z_{\max}} f(z)\, dz = \int_0^1 f(T^{-1}(y)) \cdot \frac{dT^{-1}(y)}{dy}\, dy. \tag{6.35}$$

In the ideal case mapping T makes the integrand have zero variation, that is constant $\mathcal{C}$:

$$f(T^{-1}(y)) \cdot \frac{dT^{-1}(y)}{dy} = \mathcal{C}.$$

From this we can have

$$T(z) = \frac{1}{\mathcal{C}} \cdot \int_{z_{\min}}^{z} f(Z)\, dZ.$$

Since mapping T is expected to map to $[0,1]$, we require that $T(z_{\max}) = 1$. Thus the constant $\mathcal{C}$ should be equal to $\int_{z_{\min}}^{z_{\max}} f(Z)\, dZ$. Summarizing, the uniformly distributed point y should be transformed by the inverse of the following function

$$T(z) = \frac{\int_{z_{\min}}^{z} f(Z)\, dZ}{\int_{z_{\min}}^{z_{\max}} f(Z)\, dZ}. \tag{6.36}$$

Note that this is the same transformation as has been derived for the random samples. It means that the method of importance sampling is independent of whether random samples are transformed in Monte-Carlo quadrature, or deterministic samples are transformed in quasi-Monte Carlo integration.

6.3.3 Metropolis sampling

Importance sampling requires a probability density that is proportional to a function g which, in turn, should mimic the integrand. In the previous section we discussed a random variable transformation method that can generate samples with the required probability density. This method expects function g to be symbolically integrable and its primitive function to be invertible. This requirement often contradicts the requirement of the good approximation of the original integrand.

This section discusses a different sampling strategy, proposed by Metropolis et. al [MRR$^+$53], that has less expectations towards function g. In fact, it only supposes that the integral of g can be determined either symbolically or numerically.

The Metropolis method carries out sampling by establishing a discrete time Markov process $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_i, \ldots\}$ in the space of samples, whose limiting distribution is proportional to the selected function. A discrete time Markov process visits states which correspond to samples. The Metropolis algorithm constructs this process such that the probability of being in a given state converges to a limiting value and in the limiting case $g(\mathbf{z}) = b \cdot p(\mathbf{z})$, where $b = \int_V g(\mathbf{z})\, d\mathbf{z}$.

A Markov process can be defined by the state transition probabilities, that is by the conditional probability of the next state provided that the current state is known. In Metropolis method, the next state $\mathbf{z}_{i+1}$ of this process is found by letting an almost arbitrary **tentative transition function** $T(\mathbf{z}_i \to \mathbf{z}_t)$ generate a **tentative sample** $\mathbf{z}_t$ which is either accepted as the real next state or rejected making the next state equal to the actual state using an **acceptance probability** $a(\mathbf{z}_i \to \mathbf{z}_t)$. Thus the state transition probability density from state $\mathbf{x}$ to a different state $\mathbf{y}$ is:

$$P(\mathbf{x} \to \mathbf{y})\, d\mathbf{y} = T(\mathbf{x} \to \mathbf{y})\, d\mathbf{y} \cdot a(\mathbf{x} \to \mathbf{y}). \tag{6.37}$$

The event that the process remains in the same state is the complement of moving to any other state, thus the probability of no state transition happening is:

$$1 - \int\limits_{\mathbf{x} \in V, \mathbf{x} \neq \mathbf{y}} P(\mathbf{x} \to \mathbf{y})\, d\mathbf{y} = 1 - \int\limits_{\mathbf{x} \in V, \mathbf{x} \neq \mathbf{y}} T(\mathbf{x} \to \mathbf{y}) \cdot a(\mathbf{x} \to \mathbf{y})\, d\mathbf{y}. \tag{6.38}$$

Let us denote the probability density of being in state $\mathbf{x}$ at step n by $p_n(\mathbf{x})$. Using the total probability theorem and taking advantage that in Markov processes the future depends only on the present state and is independent of the past, the following recursion can be established for these state probabilities:

$$p_{n+1}(\mathbf{y}) = \int\limits_{\mathbf{x} \in V, \mathbf{x} \neq \mathbf{y}} p_n(\mathbf{x}) \cdot P(\mathbf{x} \to \mathbf{y})\, d\mathbf{x} + \left(1 - \int\limits_{\mathbf{x} \neq \mathbf{y}} P(\mathbf{y} \to \mathbf{x})\, d\mathbf{x}\right) \cdot p_n(\mathbf{y}). \tag{6.39}$$

If the limiting probability $p(\mathbf{y}) = \lim\limits_{n \to \infty} p_n(\mathbf{y})$ exists, then it should be the fixed point of this recursion, that is:

$$p(\mathbf{y}) = p(\mathbf{y}) + \int\limits_{\mathbf{x} \in V, \mathbf{x} \neq \mathbf{y}} p(\mathbf{x}) \cdot P(\mathbf{x} \to \mathbf{y}) - p(\mathbf{y}) \cdot P(\mathbf{y} \to \mathbf{x})\, d\mathbf{x} \tag{6.40}$$

If the Markov process is ergodic, then the limiting distribution is unambigous and is independent of the initial state of the process. The process is ergodic if after a given number of steps any state can be reached from any other state with non-zero probability.

The core idea of Metropolis is to construct acceptance probability $a(\mathbf{x} \to \mathbf{y})$ in such a way that the limiting probability of the process will be $p(\mathbf{z}) = g(\mathbf{z})/b$. Substituting this goal into equation (6.40) we get:

$$g(\mathbf{y}) = g(\mathbf{y}) + \int\limits_{V, \mathbf{x} \neq \mathbf{y}} g(\mathbf{x}) \cdot P(\mathbf{x} \to \mathbf{y}) - g(\mathbf{y}) \cdot P(\mathbf{y} \to \mathbf{x})\, d\mathbf{x}. \tag{6.41}$$

This holds when the total incoming and outgoing flows of state $\mathbf{x}$ are balanced. One way of ensuring this is to require that:

$$g(\mathbf{x}) \cdot P(\mathbf{x} \to \mathbf{y}) = g(\mathbf{y}) \cdot P(\mathbf{y} \to \mathbf{x}). \tag{6.42}$$

This condition — that is also called as the **detailed balance** — means that the transitions between any two states are balanced. Using the formulae (6.37) for state transition probabilities, we can further obtain:

$$g(\mathbf{x}) \cdot T(\mathbf{x} \to \mathbf{y}) \cdot a(\mathbf{x} \to \mathbf{y}) = g(\mathbf{y}) \cdot T(\mathbf{y} \to \mathbf{x}) \cdot a(\mathbf{y} \to \mathbf{x}). \tag{6.43}$$

Thus the required ratio of the two acceptance probabilities is:

$$\frac{a(\mathbf{y} \to \mathbf{x})}{a(\mathbf{x} \to \mathbf{y})} = \frac{g(\mathbf{x}) \cdot T(\mathbf{x} \to \mathbf{y})}{g(\mathbf{y}) \cdot T(\mathbf{y} \to \mathbf{x})}. \tag{6.44}$$

Any acceptance probability satisfying this requirement makes the limiting distribution proportional to g. Considering the speed of convergence, on the other hand, the state transition probabilities and consequently the acceptance probabilities should be large. Since a probability cannot exceed 1, the optimal acceptance probability is:

$$a(\mathbf{x} \to \mathbf{y}) = \min\left\{ \frac{g(\mathbf{y}) \cdot T(\mathbf{y} \to \mathbf{x})}{g(\mathbf{x}) \cdot T(\mathbf{x} \to \mathbf{y})},\ 1 \right\}.$$

The algorithm generating a trajectory of the Markov process to obtain samples $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N\}$ is as follows:

```
for $i = 1$ to $N$ do
    Based on the actual state $\mathbf{z}_i$, choose another random, tentative state $\mathbf{z}_t$ using $T(\mathbf{z}_i \to \mathbf{z}_t)$
    $a(\mathbf{z}_i \to \mathbf{z}_t) = (g(\mathbf{z}_t) \cdot T(\mathbf{z}_t \to \mathbf{z}_i))/(g(\mathbf{z}_i) \cdot T(\mathbf{z}_i \to \mathbf{z}_t))$
    if $a(\mathbf{z}_i \to \mathbf{z}_t) \geq 1$ then $\mathbf{z}_{i+1} = \mathbf{z}_t$
    else                                        // accept with probability $a(\mathbf{z}_i \to \mathbf{z}_t)$
        Generate uniformly distributed random number $r$ in $[0, 1]$.
        if $r < a(\mathbf{z}_i \to \mathbf{z}_t)$ then $\mathbf{z}_{i+1} = \mathbf{z}_t$
        else                      $\mathbf{z}_{i+1} = \mathbf{z}_i$
    endif
endfor
```

6.3.4 Application of the VEGAS algorithm

If neither the integrand nor its approximation are available explicitly, no probability density can be constructed before starting the generation of the samples. However, as the sampling process goes on, the history of previous samples can provide information which the important regions are, thus a probability density can be built adaptively using the information gathered previously. This means that as the samples are evaluated not only the quadrature is computed but a probability density is also approximated which can be proportional to the current estimate of the integrand.

This is quite straightforward, but also generates a problem. The estimated probability density is also a high-dimensional function, thus its storage would need a lot of storage space.

The **VEGAS method** [Lep80] is an adaptive Monte-Carlo technique that generates a probability density for importance sampling automatically and in a separable form. The reason of the requirement of separability is that probability densities are computed from and stored in histogram tables and s number of 1-dimensional tables need much less storage space than a single s-dimensional table. Formally, let us assume that the probability density can be defined in the following product form:

$$p(z_1, z_2, \ldots, z_s) = p_1(z_1) \cdot p_2(z_2) \cdot \ldots \cdot p_s(\omega_s). \tag{6.45}$$

It can be shown [Lep80] that the optimal selection of p_1 is

$$p_1(z_1) \propto \sqrt{\int \ldots \int \frac{f^2(z_1, \ldots, z_D)}{p_2(z_2) \ldots p_s(z_s)}\, dz_2 \ldots dz_s}, \tag{6.46}$$

and similar formulae apply to $p_2, \ldots, p_s$. These $p_1, \ldots p_s$ functions can be tabulated as 1-dimensional tables. The jth element of the ith table represents the importance of the directional region where $z_i \in [\frac{j}{N}, \frac{(j+1)}{N}]$.

This immediately presents a recursive importance sampling strategy. The algorithm is decomposed into phases consisting of a number of samples. At the end of each phase weights $p_1, \ldots, p_s$ are refined, to provide a better probability density for the subsequent phase. Assuming that $p_1, \ldots, p_s$ are initially constant, a standard Monte-Carlo method is initiated, but in addition to accumulating to compute the integral, $p_1, \ldots, p_s$ are also estimated using equation (6.46).

Then for the following phase, the samples are selected according to the new p_i functions. In order to calculate a sample for z_i, for instance, a single random value is generated in the range of 0 and the sum of all elements in the table defining p_i. Then the elements of the table is retained one by one and summed to a running variable. When this running variable exceeds the random sample, then the searched region is found. The value in this region is then found by uniformly selecting a single point from the region.

Chapter 7

Random walk solution of the global illumination problem

Recall that expansion obtains the measured power in the form of an infinite Neumann series: $\mathcal{M}L = \sum_{i=0}^{\infty} \mathcal{M}\mathcal{T}^i L^e$. The terms of this infinite Neumann series have intuitive meaning as well: $\mathcal{M}\mathcal{T}^0 L^e = \mathcal{M}L^e$ comes from the emission, $\mathcal{M}\mathcal{T}^1 L^e$ comes from a single reflection, $\mathcal{M}\mathcal{T}^2 L^e$ from two reflections, etc.

7.1 Why should we use Monte-Carlo expansion methods?

Expansion techniques require the evaluation of very high-dimensional — in fact, infinite dimensional — integrals. When using classical quadrature rules for multi-dimensional integrals [PFTV92], such as for example the trapezoidal rule, in order to provide a result with a given accuracy, the number of sample points is in the order of $\mathcal{O}(N^s)$, where s is the dimension of the domain. This phenomenon is called the **dimensional core** or **dimensional explosion** and makes classical quadrature rules prohibitively expensive for higher dimensions.

However, Monte-Carlo or quasi-Monte Carlo techniques distribute the sample points simultaneously in all dimensions, thus they can avoid dimensional explosion. For example, the probabilistic error bound of Monte-Carlo integration is $\mathcal{O}(N^{-0.5})$, independently of the dimension of the domain. s-dimensional low discrepancy series [Nie92] can even achieve $\mathcal{O}(\log^s N/N) = \mathcal{O}(N^{-(1-\epsilon)})$ convergence rates for finite variation integrands.

Furthermore, classical quadrature cannot be used for infinite dimensional integrals, thus the Neumann series should be truncated after D terms. This truncation introduces a bias of order $\lambda^{D+1} \cdot ||L^e||/(1-\lambda)$, where λ is the contraction of the light transport operator. Using a Russian roulette based technique, on the other hand, Monte-Carlo methods are appropriate for even infinite dimensional integrals.

Thus we can conclude that the stochastic approach is indispensable for expansion methods.

In computer graphics the first Monte-Carlo random walk algorithm — called **distributed ray-tracing** — was proposed by Cook et al. [CPC84], which spawned to a set of variations, including **path-tracing** [Kaj86], **light-tracing** [DLW93], **bi-directional path-tracing** [LW93, VG95], **Monte-Carlo radiosity** [Shi91b, Neu95, PM95], and **two-pass methods** which combine radiosity and ray-tracing [Shi90, ZS95, WCG87].

The problem of naive generation of walks is that the probability that a shooting path finds the eye is zero for a pin-hole camera or very small if a non-zero aperture camera model is used, while the probability that a gathering random path ends in a lightsource may be very little if the lightsources are small, thus the majority of the paths do not contribute to the image at all, and their computation is simply waste of time. Note that shooting is always superior for view-independent algorithms since they do not have to face the problem of small aperture.

Thus, on the one hand, random walk must be combined with a deterministic step that forces the walk to go to the eye and to find a lightsource. On the other hand, **importance sampling** [Sob91] should be incorporated to prefer useful paths along which significant radiance is transferred.

Steps of the walk transfer the radiance or the potential in the scene. The source and destination of the transfer can be points in the case of continuous methods or patches in the case of finite-element methods. If the algorithm is such that it always selects a single source for shooting or single destination for gathering, then the method is called **local method**. On the other hand, if many sources and destinations are taken into consideration simultaneously in each transfer, then the method is called **global method** or **multi-path method** [Sbe96].

7.2 Quasi-Monte Carlo quadrature for the rendering equation

Quasi-Monte Carlo walk techniques mean that instead of generating the next direction randomly, the direction is sampled from a low-discrepancy point set. Since the low-discrepancy sequences have better asymptotic disrepancy than that of the random sequences, quasi-Monte Carlo methods are expected to provide more accurate results. However, the integrand of the rendering equation is discontinuous where the discontinuity is not aligned with the coordinate axes, thus its variation is infinite. These discontinuities are usually produced by the projected object boundaries. This property makes the Koksma-Hlawka inequality not appropriate for the error analysis and for the prediction of the convergence rates.

7.2.1 Integrating functions of unbounded variation

In this section the convergence speed is examined for functions which are generally smooth but have general discontinuities of finite "length". First the domain of the integration is assumed to be 2-dimensional, then the results will be generalized to arbitrary dimensions.

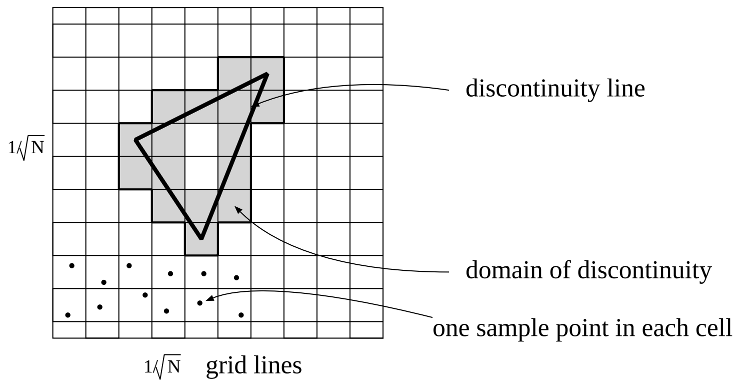


Figure 7.1: A typical integrand of the rendering equation

Suppose that N samples have been generated to estimate the integral of a function such as in figure 7.1 using a low-discrepancy sequence. In order to overcome the difficulty that the integrand f has infinite variation, the function is decomposed into two functions, one is smooth $\tilde{f}$ having continuous mixed derivatives and the other $\hat{f}$ inherits the discontinuity of f (figure 7.2).

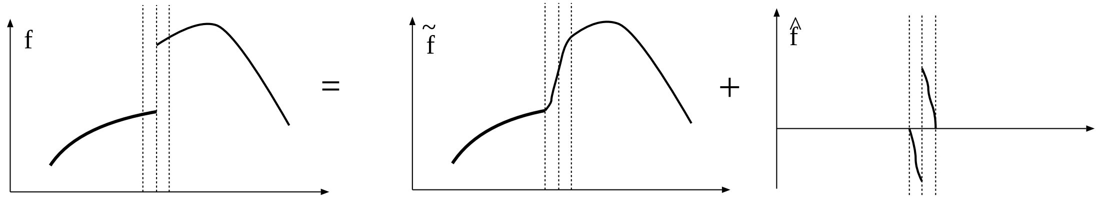


Figure 7.2: Decomposition of f into a smooth ($\tilde{f}$) and a discontinuous ($\hat{f}$) function

Low-discrepancy sequences generate points in a cellular grid in a way that the difference of the number of points in two cells is at most 1. If there are already N number of points, the size of a cell on the finest, filled level is approximately $1/\sqrt{N} \times 1/\sqrt{N}$. Let us define the domain of $\hat{f}$ as the set of those cells that are intersected by the discontinuity. This domain is called the **domain of discontinuity**. The number of such cells is in the order of the "digital length" of the discontinuity curve, which is the product of the maximum extent l and the resolution of the grid $\sqrt{N}$. Since each cell has at least 1 (and at most 2) points, the number of points in this domain is at least $l\sqrt{N}$.

The error of quadrature is as follows:

$$|\int\limits_{\mathbf{z}\in[0,1]^2} f(\mathbf{z})\,d\mathbf{z} - \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i)| \le |\int\limits_{\mathbf{z}\in[0,1]^2} \tilde{f}(\mathbf{z})\,d\mathbf{z} - \frac{1}{N}\sum_{i=1}^{N} \tilde{f}(\mathbf{z}_i)| + |\int\limits_{\mathbf{z}\in[0,1]^2} \hat{f}(\mathbf{z})\,d\mathbf{z} - \frac{1}{N}\sum_{i=1}^{N} \hat{f}(\mathbf{z}_i)|. \quad (7.1)$$

Since $\tilde{f}$ has finite variation, the first term in the error is bounded by $\mathcal{V}_{\mathrm{HK}}(\tilde{f}) \cdot \mathcal{D}^*(\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_N)$.

Concerning the second term, the integration of $\hat{f}$ is estimated taking $l\sqrt{N}$ uniformly distributed samples and averaging the result. Since the samples and the discontinuity are not related in any way, we can suppose that this is a normal Monte-Carlo integration [PFTV92]. The uniform property of low-discrepancy sequence guarantees that this pseudo-random set can be assumed to have uniform distribution. If Δf is the difference between the maximum and minimum values in the domain of discontinuity, then $\sigma^2 \leq (\Delta f)^2$. In our case the number of sample points N' is $l\sqrt{N}$ and the size of the domain V is $l/\sqrt{N}$, thus we obtain with 0.997 confidence level:

$$\int_V \hat{f}(\mathbf{z})\,d\mathbf{z} = \frac{V}{N'} \cdot \sum_{i=1}^{N'} \hat{f}(\mathbf{z}_i) \pm 3 \cdot V \cdot \frac{\Delta f}{\sqrt{N'}} = \frac{1}{N'} \cdot \sum_{i=1}^{N'} \hat{f}(\mathbf{z}_i) \pm 3 \cdot \Delta f \cdot \sqrt{l} \cdot N^{-3/4}. \tag{7.2}$$

Taking into account that $\hat{f}$ is zero outside the domain of discontinuity, equality 7.1 can be expressed as:

$$|\int_{\mathbf{z}\in[0,1]^2} f(\mathbf{z})\,d\mathbf{z} - \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i)| \leq \mathcal{V}_{\mathrm{HK}}(\tilde{f}) \cdot \mathcal{D}^*(\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_N) + 3 \cdot \Delta f \cdot \sqrt{l} \cdot N^{-3/4}. \tag{7.3}$$

For large N values the second term will be dominant, which results in $\mathcal{O}(N^{-3/4})$ error bound. This is poorer than the $\mathcal{O}(\log^2 N/N)$ bound suggested by the Koksma-Hlawka inequality assuming, for example, the application of the Halton sequence. Note that the point from where the second term dominates the first one depends on "intensity" Δf and size of the discontinuity $\sqrt{l}$. The same analysis can be carried out in higher dimensions as well. In s dimensions a discontinuity of size l would intersect $V = l \cdot N^{-1/s}$ volume of cells which would contain $N' = l \cdot N^{(s-1)/s}$ sample points. Thus the general error bound is:

$$|\int_{\mathbf{z}\in[0,1]^s} f(\mathbf{z})\,d\mathbf{z} - \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{z}_i)| \leq \mathcal{V}_{\mathrm{HK}}(\tilde{f}) \cdot \mathcal{D}^*(\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_N) + 3 \cdot \Delta f \cdot \sqrt{l} \cdot N^{-\frac{(s+1)}{2s}}. \tag{7.4}$$

Thus, for quasi-Monte Carlo integration of discontinuous functions, the order of the error bound will be in between the $\mathcal{O}(N^{-(1-\epsilon)})$ bound of finite variation functions and the $\mathcal{O}(N^{-0.5})$ bound of Monte-Carlo quadrature. The higher the dimension of the integral, the closer the Monte-Carlo and quasi-Monte Carlo techniques get in convergence speed. Thus it is still worth using quasi-Monte Carlo quadrature if the size of the discontinuity l is not very large, since in this case the error could be significantly less then for Monte-Carlo quadrature.

Numerical evidence using simple functions

In order to demonstrate the previous results, the convergences of a 2-dimensional and a 3-dimensional functions are examined, that are simple enough to analytically compute their integrals.

The 2-dimensional function is:

$$f_2(x,y) = \begin{cases} (x+y)\cdot a + 1 - 2\cdot a & \text{if } x+y > 1, \\ (x+y)\cdot a & \text{otherwise }, \end{cases} \tag{7.5}$$

where a is a free parameter in the range of $[0, 0.5]$. Note that by setting a appropriately, the intensity of the discontinuity can be controlled without altering either the value of the integral or the variation of the continuous part. If $a = 0.5$, then the function has finite variation, otherwise it has infinite variation.

The results of the simulation are shown in figure 7.3. This figure shows the maximum error after a given number of samples.

The 3-dimensional function is:

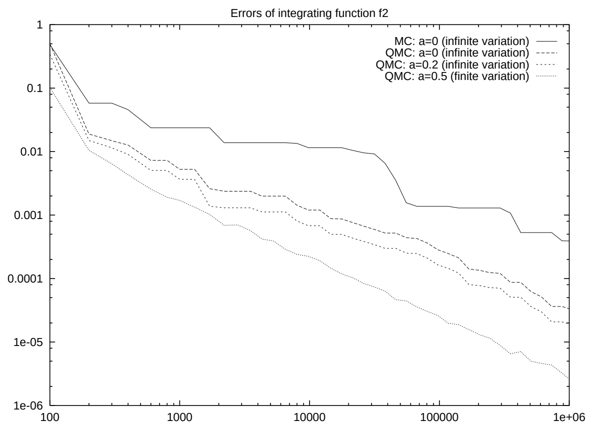
$$f_3(x,y,z) = \begin{cases} (x+y+z)\cdot a + 0.6 - 1.8\cdot a & \text{if } x+y+z > 1, \\ (x+y+z)\cdot a & \text{otherwise }, \end{cases} \tag{7.6}$$

where a is a free parameter in the range of $[0, 1/3]$. If $a = 1/3$, then f_3 has finite variation, otherwise it has not. The error of integration of f_3 is summarized in figure 7.3.

Numerical evidence for the rendering equation

The efficiency of Monte-Carlo and quasi-Monte Carlo quadratures have been tested for the presented spherical scene (section 4.4.2) assuming a single pixel camera. The error has been measured separately for the different bounces.

Looking at the error measurements of figure 7.4, we can see that even for integrands of infinite variation, quasi-Monte Carlo methods are still better but they lose their advantage when computing higher bounces as predicted by



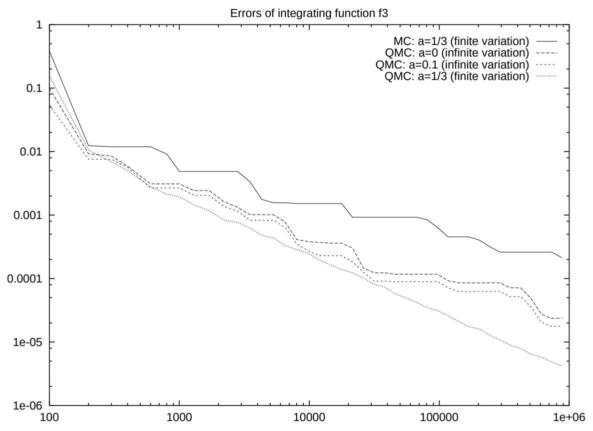
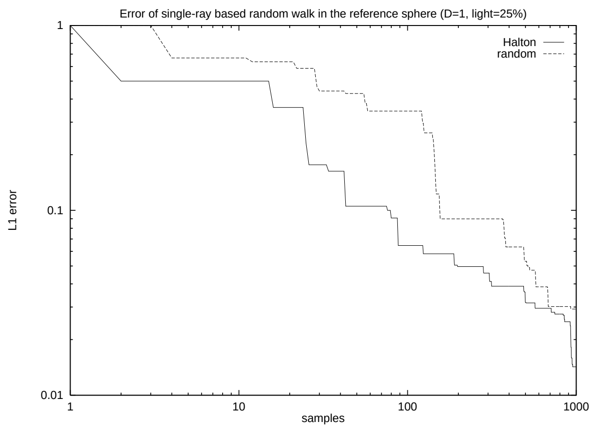


Figure 7.3: Error of integrating f_2 (left) and f_3 (right)



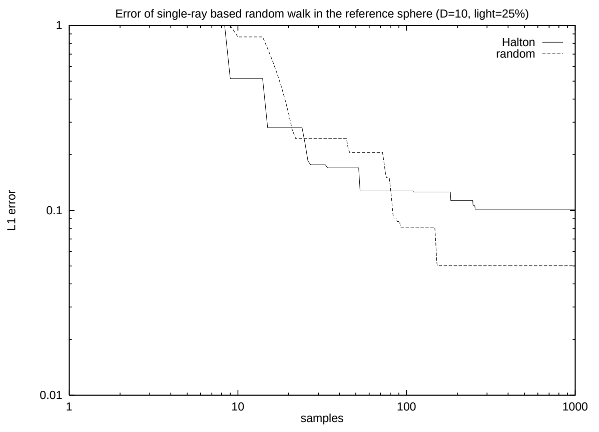


Figure 7.4: Error measurements for 1 and 10 bounces in the spherical reference scene (section 4.4.2) where the BRDF is diffuse, the albedo is 0.5, and 25 percents of the area is a diffuse lightsource

the theoretical results. The other important problem in higher dimensions is that although a low-discrepancy series has almost linearly decreasing discrepancy in the asymptotic sense, this discrepancy can still be high for not very many points (in the solution of the rendering equation we rarely use more than 1000 samples for the estimation of a single pixel). In the case of the Halton series, for example, the **base** of the series strongly affects the initial behavior of the discrepancy. These base numbers are different prime numbers for different dimensions, thus for high-dimensional integrals the base numbers can be quite high, which results in degraded performance.

7.3 Importance sampling for the rendering equation

When solving the rendering equation, usually directional integrals (or surface integrals in other formulation) should be evaluated. These directional integrals have the following form:

$$\mathcal{T}L^{\rm in}(\vec{x},\omega) = \int\limits_{\Omega} L^{\rm in}(\vec{x},\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta' \, d\omega'. \tag{7.7}$$

To allow the application of random or low-discrepancy point sets, the integration domain should be transformed to the unit cube or square. To establish such a mapping, first direction ω' is expressed by spherical coordinates ϕ, θ', which converts the directional integral to the following form:

$$\int\limits_{\Omega} L^{\rm in}(\vec{x},\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta' \, d\omega' = \int\limits_{\phi=0}^{2\pi} \int\limits_{\theta'=0}^{\pi} f_r(\phi,\theta') \cdot \cos\theta' \cdot \sin\theta' \cdot L^{\rm in} \, d\theta' d\phi \tag{7.8}$$

since $d\omega = \sin\theta' d\theta' d\phi$. Let us denote $f_r(\phi,\theta')\cdot\cos\theta'\cdot\sin\theta'$ by $w(\phi,\theta')$, which is the transfer probability density. Now we find a mapping $T(\phi,\theta') = \mathbf{z}$ that emphasizes those directions where the integrand is large and projects the domain of the spherical coordinates onto the unit square:

$$\int\limits_{\phi=0}^{2\pi} \int\limits_{\theta'=0}^{\pi} w(\phi,\theta')\cdot L^{\rm in}(\phi,\theta') \, d\theta' d\phi = \int\limits_{[0,1]^2} w(T^{-1}(\mathbf{z}))\cdot L^{\rm in}(T^{-1}(\mathbf{z}))\cdot \left| \frac{dT^{-1}(\mathbf{z})}{d\mathbf{z}} \right| d\mathbf{z} = \int\limits_{[0,1]^2} \frac{w(T^{-1}(\mathbf{z}))}{t(\mathbf{z})} \cdot L^{\rm in}(T^{-1}(\mathbf{z})) \, d\mathbf{z}, \tag{7.9}$$

where

$$\left| \frac{dT^{-1}(\mathbf{z})}{d\mathbf{z}} \right| = \frac{1}{t(\mathbf{z})}$$

is the Jacobi determinant of the inverse mapping. If the Jacobi determinant is large, then a small portion of the unit square is mapped onto a large region. Thus sample points that are uniformly distributed in the unit square will be quite rare in these regions. Alternatively, where the Jacobi determinant is small, the sample points will be dense. Considering this, the meaning of $t(\mathbf{z})$ is the **density** of sample points in the neighborhood of $\omega = (\phi,\theta') = T^{-1}(\mathbf{z})$. This has an illustrative content for the random case. If $\mathbf{z}$ is uniformly distributed random variable, then the probability density of $\omega = T^{-1}(\mathbf{z})$ will be $t(\mathbf{z})$.

The solution of the rendering equation for a given point $(\vec{x},\omega)$ requires the evaluation of the following multi-dimensional integral (equation (4.5)):

$$L(\vec{x},\omega) = L^e + \mathcal{T}L^e + \mathcal{T}^2 L^e + \ldots = \int\limits_{[0,1]^2} \ldots \int\limits_{[0,1]^2} L^e + \frac{w_1}{t_1}\cdot L^e + \frac{w_1}{t_1}\cdot\frac{w_2}{t_2}\cdot L^e + \ldots d\mathbf{z}_1 d\mathbf{z}_2 \ldots \tag{7.10}$$

This can be estimated by Monte-Carlo or quasi-Monte Carlo quadratures which evaluate the integrand in sample points and average the results. A crucial design decision of such an algorithm is the selection of mappings T_i to have good importance sampling. Using probabilistic approach, it means that the probability of selecting a walk is proportional to its contribution. Following the directions concluded from the Koksma-Hlawka inequality, the mappings should make the integrand flat — that is of low variation, or constant in the ideal case.

Looking at formula (7.10), which is the single multi-dimensional solution of the rendering equation, this decision seems to be hard to made, since there are too many free parameters to control simultaneously. Fortunately, the solution can also be presented in the following recursive form:

$$L(\vec{x},\omega) = L^e + \int\limits_{[0,1]^2} \frac{w_1}{t_1} \cdot [L^e + \int\limits_{[0,1]^2} \frac{w_2}{t_2} \cdot [L^e + \ldots] \ldots] \, d\mathbf{z}_1 d\mathbf{z}_2 \ldots \tag{7.11}$$

If we could ensure that each of the integrands of the form

$$\int_{[0,1]^2} \frac{w_i}{t_i} \cdot [L^e + \int_{[0,1]^2} \ldots] \, d\mathbf{z}_i$$

is constant (at least approximately), then the integrand of the single multi-dimensional integral will also be constant [SKCP99]. An optimal importance sampling strategy thus requires density t_i to be proportional to the product of the incoming illumination $L^e + \int \ldots$ and the cosine weighted BRDF w_i. Unfortunately, during random walks the incoming non-direct illumination is not known (the random walk is just being done to estimate it).

Thus we have to use approximations for which we have three alternatives. Firstly, information about the illumination in the space can be gathered in a preprocessing phase, then this information can be used to obtain probability densities for importance sampling. This is called the **global importance sampling**. These methods can be classified according to the data structure built in the preprocessing phase. Since the ray-space is 5-dimensional, it is straightforward to apply a **5D adaptive tree** [LW96] that is similar to the well-known octree to store radiance information. Jensen proposed the application of the **photon-map** as the basis of importance sampling [Jen95]. We assigned the power computed in the preprocessing phase to **links** established between two interacting patches [SKCP98].

The second alternative is using the information gained during previous walks to approximate the illumination. This strategy is called **adaptive importance sampling**. Adaptive importance sampling methods neither require the non-uniform probability densities to be constructed in advance, nor simplify them to take into account only local properties, but converge to a desired probability density using the knowledge of previous samples. Three techniques are particularly important, which have also been used in rendering: **genetic algorithms** [LB94] the **Metropolis sampling** [MRR+53, VG97] and the **VEGAS method** [Lep80, SK98a]. The first use of Metropolis sampling in rendering aimed at speeding up bi-directional path tracing [VG97].

In the third alternative, the problem is simplified and the indirect illumination is not considered in importance sampling. When the directions are generated, we use only transfer probability density w_i and L^e representing the direct illumination of the actual point. This is called the **local importance sampling**.

It turns out that we have to encounter severe problems when we have to find a mapping which has a density that is proportional to the product of the effects of the transfer probability density and the direct lighting. Consequently, local importance sampling strategies usually use only either w_i or L^e to identify important directions. The first alternative is called the **BRDF sampling**, while the second is called the **lightsource sampling**.

7.3.1 BRDF sampling

BRDF based importance sampling means that at step i the density t_i is proportional to the transfer probability density w_i, that is

$$t_i \propto w_i = f_r(\omega_{\text{in}}, \vec{x}, \omega_{\text{out}}) \cdot \cos\theta \sin\theta. \tag{7.12}$$

In gathering algorithms ω_{out} is known, θ is the angle between ω_{in} and the surface normal, and ω_{in} should be determined. In shooting algorithms, on the other hand, ω_{in} is known, θ is the angle between ω_{out} and the surface normal, and ω_{out} should be determined.

Due to the fact that t_i represents density (probability density for Monte-Carlo methods), its integral is 1. Thus for gathering walks and for non-transparent materials, the ratio of proportionality in equation (7.12) is

$$\int_{\Omega_H} w \, d\omega_{\text{in}} = \int_{\Omega_H} f_r(\omega_{\text{in}}, \vec{x}, \omega_{\text{out}}) \cdot \cos\theta_{\text{in}} \, d\omega_{\text{in}} = a(\vec{x}, \omega_{\text{out}})$$

where $a(\vec{x}, \omega_{\text{out}})$ is the **albedo** of the surface at point $\vec{x}$ in the outgoing direction. Similarly, the proportionality ratio for shooting walks is

$$\int_{\Omega_H} w \, d\omega_{\text{out}} = \int_{\Omega_H} f_r(\omega_{\text{in}}, \vec{x}, \omega_{\text{out}}) \cdot \cos\theta_{\text{out}} \, d\omega_{\text{out}} = a(\vec{x}, \omega_{\text{in}}).$$

Thus the weights $w^* = w_i/t_i$ are the albedos at the visited points.

BRDF sampling for diffuse materials

Diffuse materials have constant BRDF, that is

$$t_i(\phi, \theta) \propto w_i = f_r \cdot \cos\theta \sin\theta.$$

The proportionality ratio is found to make t_i to integrate to 1:

$$t_i(\phi,\theta) = \frac{w_i}{\int\limits_{\Omega_H} w_i \, d\omega} = \frac{f_r \cdot \cos\theta \sin\theta}{\int\limits_{\phi=0}^{2\pi}\int\limits_{\theta=0}^{\pi/2} f_r \cdot \cos\theta\sin\theta \, d\theta d\phi} = \frac{\cos\theta\sin\theta}{\pi}.$$

Assuming that the random variables used to produce the two coordinate samples are independent, this density is obtained in a product form:

$$t_i(\phi,\theta) = \frac{1}{2\pi} \cdot [2\cos\theta\sin\theta], \tag{7.13}$$

where $1/(2\pi)$ is the probability density of ϕ and $2\cos\theta\sin\theta = \sin 2\theta$ is the probability density of θ.

The corresponding probability distribution functions are:

$$P(\phi) = \int\limits_0^\phi \frac{1}{2\pi} \, d\Phi = \frac{\phi}{2\pi}, \qquad P(\theta) = \int\limits_0^\theta \sin 2\Theta \, d\Theta = \sin^2\theta.$$

Thus the required ϕ and θ random variables can be found by the following transformations of u, v variables that are uniformly distributed in $[0,1]$ (section 6.3.1):

$$\phi = 2\pi \cdot u, \quad \theta = \arcsin\sqrt{v}.$$

The transformed weight after importance sampling is the albedo

$$w_i^* = \frac{w_i}{t_i} = f_r \cdot \pi = a. \tag{7.14}$$

BRDF sampling for specular materials

Specular materials can be characterized by the reciprocal version of the Phong's BRDF model, that is

$$f_r(\omega_{\rm in}, \vec{x}, \omega_{\rm out}) = k_s \cdot \cos^n\psi \cdot \epsilon(\pi/2 - \theta),$$

where ψ is the angle between $\omega_{\rm out}$ and the mirror direction of $\omega_{\rm in}$ onto the surface normal, which will be referred to as ω_r, and $\epsilon(\pi/2-\theta)$ indicates that the outgoing direction cannot point into the object, i.e. the angle θ between the surface normal and the outgoing direction should be less than 90 degrees.

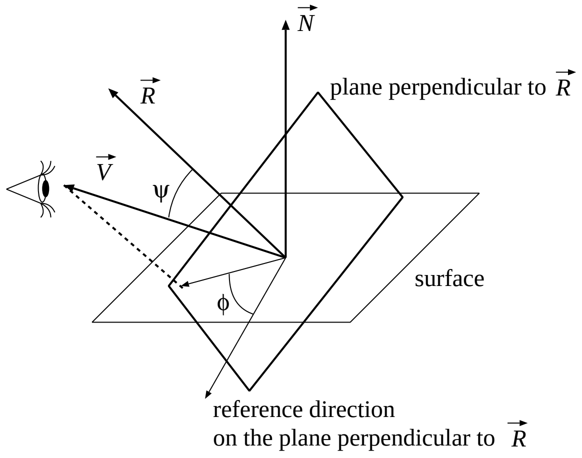

$\vec{N}$
$\vec{R}$
plane perpendicular to $\vec{R}$
$\vec{V}$
Ψ
surface
ϕ
reference direction
on the plane perpendicular to $\vec{R}$

Figure 7.5: Parameterization for the calculation of the albedo

In order to appropriately parameterize the directional sphere, now the north pole is selected by the reflection direction ω_r (figure 7.5). Let us identify a direction by an angle from ω_r, that is by ψ, and by another angle ϕ between its projection onto a plane perpendicular to ω_r and an arbitrary vector on this plane.

BRDF sampling requires a density which satisfies the following criterion:

$$t_i(\phi,\psi) \propto w_i = k_s \cdot \cos^n\psi \cdot \cos\theta(\psi,\phi) \cdot \epsilon(\pi/2 - \theta(\psi,\phi)) \cdot \sin\psi.$$

Unfortunately, the $\cos\theta \cdot \epsilon(\pi/2-\theta)$ factor forbids the symbolic integration of this formula, thus we will use a density that is proportional only to $\tilde{w}_i = k_s \cdot \cos^n\psi\sin\psi$. The proportionality ratio is found to make t_i to integrate to 1:

$$t_i(\phi,\psi) = \frac{k_s \cdot \cos^n\psi\sin\psi}{\int\limits_{\phi=0}^{2\pi}\int\limits_{\psi=0}^{\pi/2} k_s \cdot \cos^n\psi\sin\psi\; d\psi d\phi} = \frac{n+1}{2\pi}\cos^n\psi\sin\psi.$$

Assuming that the random variables used for producing the two coordinate samples are independent, this density is obtained in a product form:

$$t_i(\phi,\psi) = \frac{1}{2\pi}\cdot[(n+1)\cos^n\psi\sin\psi], \tag{7.15}$$

where $1/(2\pi)$ is the probability density of ϕ and $(n+1)\cos^n\psi\sin\psi$ is the probability density of ψ.

The corresponding probability distribution functions are:

$$P(\phi) = \frac{\phi}{2\pi}, \qquad P(\psi) = \int\limits_0^{\psi}(n+1)\cos^n\Psi\sin\Psi\; d\Psi = 1-\cos^{n+1}\psi.$$

Thus the required ϕ and θ random variables can be found by the following transformations of u, v variables that are uniformly distributed in $[0,1]$:

$$\phi = 2\pi\cdot u, \qquad \psi = \arccos(1-v)^{1/(n+1)}.$$

The transformed weight after importance sampling is

$$w_i^* = \frac{w_i}{t_i} = \frac{2\pi k_s}{n+1}\cdot\cos\theta(\psi,\phi)\cdot\epsilon(\pi/2-\theta(\psi,\phi)). \tag{7.16}$$

Different other specular BRDF models are presented and their BRDF sampling is discussed in [War92, NNSK98b, NNSK98a, NNSK99a].

7.3.2 Lightsource sampling

Lightsource sampling is used in **direct lightsource calculations** [SWZ96] and as a complementary sampling strategy to BRDF sampling in random walks.

Since in this case, the samples are selected from a lightsource instead of the directional sphere, the surface integral form of the transport operator is needed:

$$(\mathcal{T}L^e)(\vec{x},\omega) = \int\limits_{\Omega} L^e(h(\vec{x},-\omega'),\omega')\cdot f_r(\omega',\vec{x},\omega)\cdot\cos\theta'\; d\omega' = \int\limits_{S_e} L^e(\vec{y},\omega_{\vec{y}\to\vec{x}})\cdot f_r(\omega_{\vec{y}\to\vec{x}},\vec{x},\omega)\cdot\frac{\cos\theta'_{\vec{x}}\cdot\cos\theta_{\vec{y}}}{|\vec{x}-\vec{y}|^2}\cdot v(\vec{y},\vec{x})\; d\vec{y}, \tag{7.17}$$

where $v(\vec{y},\vec{x})$ is 1 if points $\vec{x}$ and $\vec{y}$ are not occluded from each other and 0 otherwise, and S_e is the surface of non-zero emission. To obtain a Monte-Carlo estimate for this integral, N points $\vec{y}_1,\ldots\vec{y}_N$ are sampled uniformly on the lightsource and the following formula is used:

$$(\mathcal{T}L^e)(\vec{x},\omega) \approx \frac{|S_e|}{N}\cdot\sum_{i=1}^{N} L^e(\vec{y}_i,\omega_{\vec{y}_i\to\vec{x}})\cdot v(\vec{y}_i,\vec{x})\cdot f_r(\omega_{\vec{y}_i\to\vec{x}},\vec{x},\omega)\cdot\frac{\cos\theta'_i\cdot\cos\theta_{\vec{y}_i}}{|\vec{x}-\vec{y}_i|^2}. \tag{7.18}$$

If the scene has a single homogeneous lightsource which is relatively small and is far from the considered point, then the integrand will be approximately constant on the lightsource surface, thus this estimator has low variance.

7.3.3 Sampling the lightsources in gathering random walks

Since lightsource sampling generates samples only on the direct lightsources, it completely ignores indirect illumination. Thus it cannot be used alone in global illumination algorithms, but only as a complementary part of, for example, BRDF sampling.

The simplest way to combine the two strategies is to generate all but the last directions of the gathering walk by sampling the BRDF and to compute the last direction by sampling the lightsource. Note that when stopping the walk, the indirect illumination is assumed to be zero, thus following the directions of the lightsources is a reasonable approach.

Another combination strategy is to trace one or more shadow rays from each visited point of the walk towards the lightsources, not only from the last of them.

Formally, this approach can be presented as a restructuring of the Neumann series

$$L = L^e + \mathcal{T}L^e + \mathcal{T}^2 L^e + \mathcal{T}^3 L^e \ldots = L^e + (\mathcal{T}L^e) + \mathcal{T}(\mathcal{T}L^e) + \mathcal{T}^2(\mathcal{T}L^e) \ldots \tag{7.19}$$

and using lightsource sampling for the $L^{e*} = (\mathcal{T}L^e)$ integral while sampling the BRDFs when evaluating the $\mathcal{T}^i L^{e*}$ integrals. Practically it means that having hit a surface, one or more shadow rays are traced towards the lightsources and the reflection of the illumination of this point is estimated. This reflection is used as if it were the emission of the surface. This method is particularly efficient if the scene consists of point lightsources. Tracing a single ray to each point lightsource, the illumination due to the point lightsources can be determined exactly (with zero variance).

If the scene contains small and large lightsources, then lightsource sampling can be used only for the smaller lightsources. Formally, the unknown radiance L is decomposed into two terms:

$$L = L^{ep} + L^{np} \tag{7.20}$$

where L^{ep} is the emission of the small, point-like lightsources, L^{np} is the emission of the large area lightsources and the reflected radiance. Substituting this into the rendering equation we have:

$$L^{ep} + L^{np} = L^e + \mathcal{T}(L^{ep} + L^{np}). \tag{7.21}$$

Expressing L^{np} we obtain:

$$L^{np} = (L^e - L^{ep} + \mathcal{T}L^{ep}) + \mathcal{T}L^{np}. \tag{7.22}$$

Introducing the new lightsource term

$$L^{e*} = L^e - L^{ep} + \mathcal{T}L^{ep} \tag{7.23}$$

which just replaces the point lightsources (L^{ep}) by their effect ($\mathcal{T}L^{ep}$), the equation for L^{np} is similar to the original rendering equation:

$$L^{np} = L^{e*} + \mathcal{T}L^{np}. \tag{7.24}$$

When this is solved, small lightsources are replaced by their single reflection, then the solution is added to the radiance of the small lightsources (equation 7.20).

7.3.4 Importance sampling in colored scenes

So far, we have assumed that the weights containing the BRDFs and the emissions are scalars thus the densities can be made proportional to them. This is only true if the rendering equation is solved on a single wavelength.

However, if color images are needed, the rendering equation should be solved on several (at least 3) different wavelengths. If the different wavelengths are handled completely independently, then the proposed importance sampling strategy can be used without any modifications. However, this approach carries out geometric calculations, such as tracing rays, independently and redundantly for different wavelengths, thus it cannot be recommended.

A better approach is using rays that transport light on all wavelengths simultaneously. In this case the emission and the BRDF can be represented by vectors, thus to allow importance sampling, we need a scalar **importance function** $\mathcal{I}$ that is large when the elements in the vector are large and small when the elements are small. The importance is a functional of the spectrum. A straightforward way is using the **luminance** of the spectrum since it emphasizes those wavelengths to which the eye is more sensitive.

7.3.5 Multiple importance sampling

So far, we mentioned two basic importance sampling strategies, the BRDF sampling and the lightsource sampling, which are local in the sense that they focus on a single reflection. It is easy to imagine that if the sampling considers simultaneously many reflections, then the number of possible strategies increases dramatically.

Obviously, we desire to use the best sampling strategy. Unfortunately the performance of a sampling strategy depends on the properties of the scene, which is usually not known a-priori, thus the best strategy cannot be selected. Instead of selecting the best, Veach and Guibas [VG95] proposed to combine several strategies in a way that the strengths of the individual sampling methods are preserved.

Suppose that we can use n different sampling techniques for generating random paths, where the distribution of the samples is constructed from several $p_1, ..., p_n$ importance sampling distributions. The number of samples taken from p_i is denoted by M_i, and the total number of samples by $M = \sum_i M_i$. The M_i values are fixed in advance before any samples are taken. The "average probability density" of selecting the sample $\mathbf{z}$ is then

$$\hat{p}(\mathbf{z}) = \sum_{i=1}^{n} \frac{M_i}{M} \cdot p_i(\mathbf{z}). \tag{7.25}$$

Thus the integral quadrature using these samples is

$$\int\limits_{[0,1]^s} f(\mathbf{z})\, d\mathbf{z} = \int\limits_{[0,1]^s} \frac{f(\mathbf{z})}{\hat{p}(\mathbf{z})} \cdot \hat{p}(\mathbf{z})\, d\mathbf{z} \approx \frac{1}{M} \sum_{i=1}^{n} \sum_{j=1}^{M_i} \frac{f(\mathbf{z}_{i,j})}{\hat{p}(\mathbf{z}_{i,j})} = \sum_{i=1}^{n} \frac{1}{M_i} \sum_{j=1}^{M_i} w_i(\mathbf{z}_{i,j}) \cdot \frac{f(\mathbf{z}_{i,j})}{p_i(\mathbf{z}_{i,j})} \tag{7.26}$$

where $\mathbf{z}_{i,j}$ is the jth sample taken from the ith distribution, and the weights are

$$w_i(\mathbf{z}) = \frac{M_i \cdot p_i(\mathbf{z})}{\sum_{k=1}^{n} M_k \cdot p_k(\mathbf{z})}. \tag{7.27}$$

Let us interpret this result when all methods use the same number of samples. $p_i(\mathbf{z})$ is the probability that a sample $\mathbf{z}$ is generated by method i. The samples are combined with this weight which guarantees that no sample will be accounted for twices. In order to have an unbiased estimation, $\sum_i w_i(\mathbf{z}) = 1$ should hold for all $\mathbf{z}$.

7.4 Handling infinite-dimensional integrals

Expansion, or walk methods require the evaluation of a series of integrals where the dimension goes to infinity. One way of attacking the problem is truncating the Neumann series, but this introduces some bias, which can be quite high if the scene is highly reflective. Fortunately, there is another approach that solves the infinite-dimensional integration problem through randomization. In the context of Monte-Carlo integration, this approach is called the **Russian roulette** [AK90], but here a somewhat more general treatment is also given that can also justify this approach for quasi-Monte Carlo quadratures.

7.4.1 Russian roulette

Note that the Neumann series contains a sequence of the following integrals:

$$L = \int\limits_{[0,1]^2} \frac{w}{t} \cdot [L^e + \ldots]\, d\mathbf{z} = \int\limits_{[0,1]^2} w^*(\mathbf{z}) \cdot L^{\text{in}}(\mathbf{z})\, d\mathbf{z} = E\left[w^* \cdot L^{\text{in}}\right].$$

A Monte-Carlo quadrature would generate random samples in the domain and estimate the integral as an average of the integrand at these samples. Let us further randomize this computation and before each sample let us decide randomly with probability s whether we really evaluate the integrand at the sample point or simply assume that the integrand is zero without any calculations. In order to compensate the not computed terms, when the integrand is really computed, it is divided by probability s. This randomization introduces a new random variable L^{ref} which is equal to $w^* \cdot L^{\text{in}}/s$ if the integrand is evaluated and zero otherwise. The Monte-Carlo quadrature, which provides the estimate as an expected value will still be correct:

$$E[L^{\text{ref}}] = s \cdot E\left[L^{\text{ref}} \mid \text{sample is used}\right] + (1-s) \cdot E\left[L^{\text{ref}} \mid \text{sample is not used}\right] =$$

$$s \cdot E\left[\frac{w^* \cdot L^{\text{in}}}{s}\right] + (1-s) \cdot 0 = E\left[w^* \cdot L^{\text{in}}\right] = L. \tag{7.28}$$

The variance of the new estimator, on the other hand, is increased:

$$D^2[L^{\text{ref}}] = E[(L^{\text{ref}})^2] - E^2[L^{\text{ref}}] = s \cdot E\left[\left(\frac{L^{\text{ref}}}{s}\right)^2\right] + (1-s) \cdot 0 - E^2[L^{\text{ref}}] =$$

$$\left(\frac{1}{s} - 1\right) \cdot E[(w^* \cdot L^{\text{in}})^2] + D^2[w^* \cdot L^{\text{in}}]. \tag{7.29}$$

7.4.2 Russian roulette in quasi-Monte Carlo quadrature

In the context of quasi-Monte Carlo integration Russian roulette should be explained differently because there is no “randomization” in deterministic techniques. This discussion is also used to generalize the basic concept and to show that termination decision can be made using the complete previous path not only the actual state.

Instead of randomization, we can suppose that the domain of integration is extended by additional variables on which a contribution indicator function is defined that will determine whether or not a higher order term is included in the quadrature (interestingly, in order to get rid of high-dimensional integrals, we increase the dimension of the integration). In order to compensate the missing terms in the integral quadrature, the really computed terms are multiplied by an appropriate factor. If the used contribution indicator is such that the domain where it is non-zero shrinks quickly, then the possibility of requiring very high dimensional integrals is rather low, which saves computation time. However, the integral quadrature will still be correct asymptotically.

A term of the Neumann series has generally the following form

$$L = \int \ldots \int W(\mathbf{z}_1, \ldots, \mathbf{z}_n) \cdot L^e(\mathbf{z}_1, \ldots \mathbf{z}_n)\, d\mathbf{z}_1 \ldots d\mathbf{z}_n, \tag{7.30}$$

where $W(\mathbf{z}_1, \ldots, \mathbf{z}_n)$ is the product of the weights $w_1(\mathbf{z}_1) \cdot \ldots \cdot w_n(\mathbf{z}_n)$ including the cosine functions of the angles and the BRDFs, or the product of the ratios of weights and densities $w_1^* \cdot \ldots \cdot w_n^* = w_1/t_1 \cdot \ldots \cdot w_n/t_n$, depending whether or not BRDF sampling has already been applied.

Let us extend this by a **contribution indicator function** $C(\mathbf{z}_1, r_1, \ldots \mathbf{z}_n, r_n)$ that is constant 1 if a sample $\mathbf{z}_1, \ldots \mathbf{z}_n$ should be taken into account in the integral quadrature and 0 if it should not and its contribution is assumed to be zero. Usually this function is in separable form

$$C(\mathbf{z}_1, r_1, \ldots \mathbf{z}_n, r_n) = \prod_{i=1}^{n} c_i(\mathbf{z}_i, r_i),$$

where $c_i(\mathbf{z}_i, r_i) = 1$ means that the walk must be continued at step i and $c_i(\mathbf{z}_i, r_i) = 0$ forces the walk to stop. Function $c_i(\mathbf{z}_i, r_i)$ can be defined, for instance, by a new weight function $\xi(\mathbf{z}_i)$ in the following way:

$$c_i(\mathbf{z}_i, r_i) = \begin{cases} 1 & \text{if } \xi(\mathbf{z}_i) > r_i, \\ 0 & \text{otherwise.} \end{cases}$$

The “possibility” of really computing a walk $\mathbf{z}_1, \ldots \mathbf{z}_n$ is

$$P(\mathbf{z}_1, \ldots \mathbf{z}_n) = \int\limits_{r_1=0}^{1} \ldots \int\limits_{r_n=0}^{1} C(\mathbf{z}_1, r_1, \ldots \mathbf{z}_n, r_n)\, dr_1 \ldots dr_n.$$

We can define the following function of variables $r_1, \ldots, r_n$,

$$L_r(r_1, \ldots, r_n) = \int \ldots \int C(\mathbf{z}_1, r_1, \ldots \mathbf{z}_n, r_n) \cdot \tilde{W} \cdot \tilde{L}^e\, d\mathbf{z}_1 \ldots d\mathbf{z}_n, \tag{7.31}$$

where $\tilde{W}$ and $\tilde{L}^e$ are appropriate modifications of W and L^e, which can compensate the missing terms.

The integral of this function is

$$\int\limits_{r_1=0}^{1} \ldots \int\limits_{r_n=0}^{1} I_n(r_1, \ldots, r_n)\, dr_1 \ldots dr_n = \int\limits_{r_1=0}^{1} \ldots \int\limits_{r_n=0}^{1} \int \ldots \int C(\mathbf{z}_1, r_1, \ldots \mathbf{z}_n, r_n) \cdot \tilde{W} \cdot \tilde{L}^e\, d\mathbf{z}_1 \ldots d\mathbf{z}_n\, dr_1 \ldots dr_n =$$

$$\int \ldots \int P(\mathbf{z}_1, \ldots, \mathbf{z}_n) \cdot \tilde{W} \cdot \tilde{L}^e\, d\mathbf{z}_1 \ldots d\mathbf{z}_n. \tag{7.32}$$

A sufficient requirement for this integral to be equal to the original integral L is

$$P(\mathbf{z}_1, \ldots, \mathbf{z}_n) \cdot \tilde{W} \cdot \tilde{L}^e = W \cdot L^e. \tag{7.33}$$

There are many possible selections of the contribution indicator and the $\tilde{W}$ and $\tilde{L}^e$ functions, that can satisfy this requirement, thus there are many different unbiased estimators. A widely used selection is letting

$$\tilde{W} = 1, \quad \tilde{L}^e = L^e \;\text{ and }\; \xi_i(\mathbf{z}_i) = w_i(\mathbf{z}_i).$$

which corresponds to continuing the walk after step i only if $w(\mathbf{z}_i) > r_i$. If importance sampling has already been applied, then the walk is continued if $w^* > r_i$. Since w^* approximates the albedo, this strategy continues the walk with the probability of the albedo.

BRDF sampling for materials of multiple reflection type

Practical reflection models incorporate different simple BRDFs. For example, a lot of materials can be well modeled by a sum of diffuse and specular reflections. So far, methods have been presented that are good for either the diffuse or the specular reflection, but not for the sum of them.

Fortunately, Russian-roulette can also be extended to handle these cases. If the reflection model is a sum of different BRDFs, then a random selection can be made from the different components. Suppose that the transfer probability density is available in the form of a sum of the weights corresponding to elementary BRDFs:

$$w = w_1 + w_2 + \ldots + w_n.$$

Thus the radiance of a single reflection is:

$$L = \int_{\Omega} w \cdot L^{\text{in}} \, d\omega = \int_{\Omega} w_1 \cdot L^{\text{in}} \, d\omega + \ldots + \int_{\Omega} w_n \cdot L^{\text{in}} \, d\omega.$$

Suppose that mappings t_i can be found for each integral, that also mimics important directions:

$$L = \int_{[0,1]^2} \frac{w_1}{t_1} \cdot L^{\text{in}} \, d\mathbf{z} + \ldots + \int_{[0,1]^2} \frac{w_n}{t_n} \cdot L^{\text{in}} \, d\mathbf{z} = E\left[w_1^* \cdot L^{\text{in}}\right] + \ldots + E\left[w_n^* \cdot L^{\text{in}}\right].$$

Let us select the ith BRDF with probability p_i and weight the resulting radiance by $1/p_i$ or stop the walk with probability $p_0 = 1 - p_1 - \ldots - p_n$. Thus the new random variable $\hat{L}$ is $w_i^* \cdot L^{\text{in}}/p_i$ if the ith model is used, and 0 if no model is selected. The expected value of $\hat{L}$ will still be correct:

$$E[\hat{L}] = p_1 \cdot E\left[\frac{w_1^* \cdot L^{\text{in}}}{p_1}\right] + \ldots + p_n \cdot E\left[\frac{w_n^* \cdot L^{\text{in}}}{p_n}\right] + (1 - p_1 - \ldots - p_n) \cdot 0 = E\left[(w_1^* + \ldots + w_n^*) L^{\text{in}}\right] = L. \quad (7.34)$$

This is a Monte-Carlo estimation of the sum. According to importance sampling, the variance will be small if $w_i^* L^{\text{in}}/p_i$ can be made nearly constant. Since we usually do not have a-priori information about L^{in}, w_i^*/p_i can be made a constant number. Thus to obtain a low-variance estimator, an elementary BRDF should be selected with the probability of its transformed weight w_i^*. Note that the weight w_i^* may either be equal or approximate the albedo, thus a low-variance estimator selects an elementary BRDF with the probability of its albedo.

In order to generate an "out" direction from the "in" direction and the surface normal, the following general BRDF sampling algorithm can be used:

```
BRDFSampling(in, normal, out)
    prob = SelectBRDFModel(normal, in)
    if prob = 0 then return 0
    prob *= Reflection(in, normal, out)
    if prob = 0 then return 0
    return prob
end
```

In this program "SelectBRDFModel" randomly selects an elementary BRDF from those that compose the given BRDF with a probability which approximates the albedo and also returns the selection probability. If it returns 0, then it has decided that the walk has to be stopped because of Russian-roulette. "Reflection" generates a new direction "out" with a probability density that is approximately proportional to the transfer probability density of the selected reflection model and returns the selection probability.

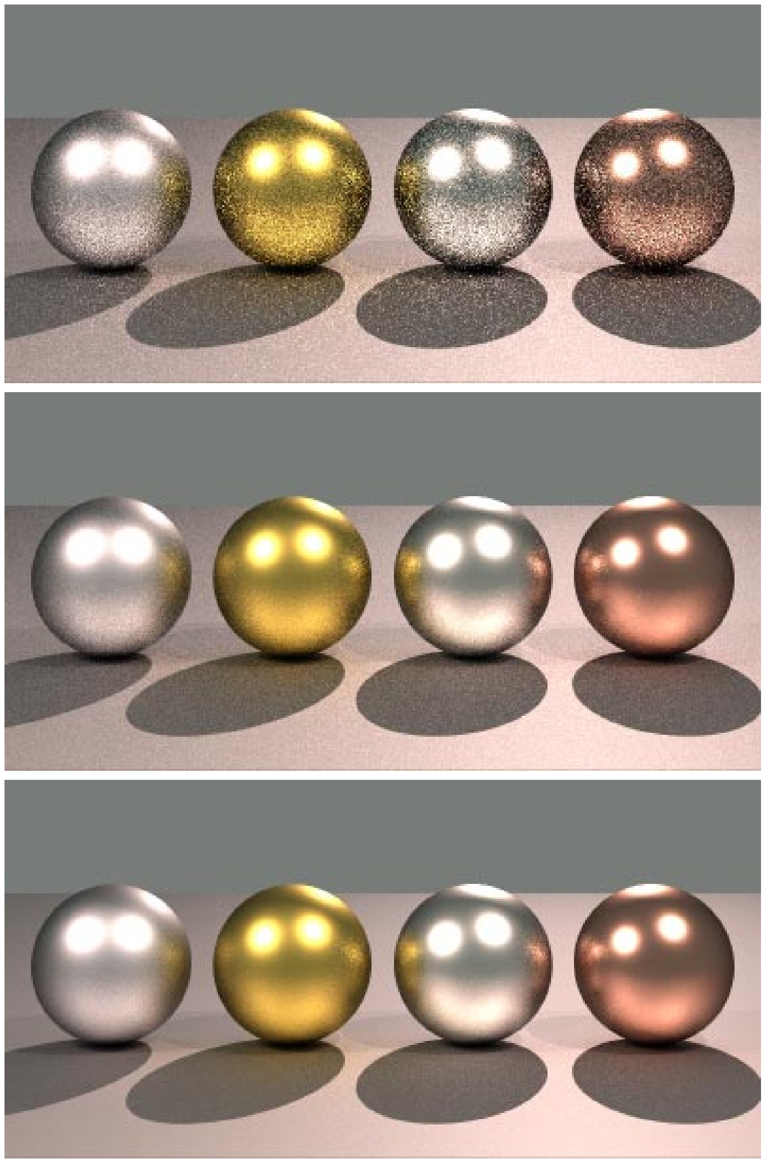


Figure 7.6: Metallic spheres generated by path tracing with 50 samples per pixel (top: importance sampling according to the cosine angle only (51 min); middle: Russian roulette based importance sampling (46 min); bottom: normal importance sampling (54 min)

Chapter 8

Review of random walk algorithms

In this chapter a number of practical random walk algorithms are reviewed and analyzed. For completeness, non-random and non global illumination methods, such as ray-casting and recursive ray-tracing are also included. The primary classification of random walk algorithms is based on the direction of generated rays. If the walks are started at the eye and go opposite to the light, then the algorithm is called **gathering**. On the other hand, if the walks originate at the lightsources and go in the direction of the light, then the algorithm is called **shooting**.

8.1 Gathering-type random walk algorithms

Gathering type random walks correspond to the Monte-Carlo solution of the rendering equations. They start at the eye position and gather the emission of the visited points.

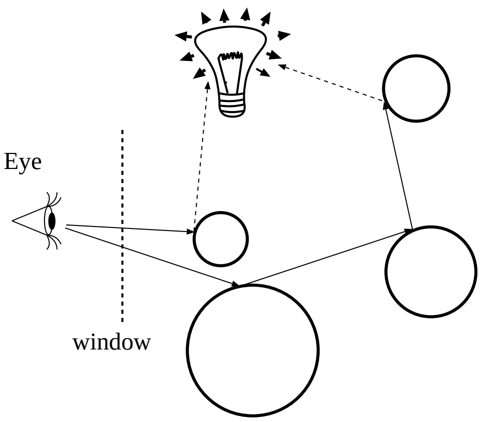


Figure 8.1: Gathering-type random walks

The general structure of gathering algorithms is as follows:

```
for each pixel $p$ do
    color = 0
    for $i$ = 1 to $N$ do
        ray = sample ray randomly from the eye through pixel $p$
        samplecolor = $c \cdot$ Trace( ray )
        color += samplecolor/$N$
    endfor
    write($p$, color)
endfor
```

In different gathering algorithms the “Trace” function is implemented differently. This function returns the radiance carried by this ray to the eye. The radiance is then multiplied by value $c = (c/S_p) \cdot S_p$ where c/S_p scaling is required by the measuring function (equation (2.39)) and S_p is the size of the integration domain.

8.1.1 Ray-casting

Ray-casting is a local-illumination algorithm of type LDE, which replaces the unknown radiance inside the integral of the rendering equation by an approximation of the emission function. In its trace function the following simplification is used to determine the radiance of a point:

$$L(\vec{x},\omega) = L^e(\vec{x},\omega) + \int_{\Omega} L_{\text{lightsource}}(h(\vec{x},-\omega'),\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta'\, d\omega', \tag{8.1}$$

where $L_{\text{lightsource}}$ is known a-priori and may be a simplification of the emission function L^e.

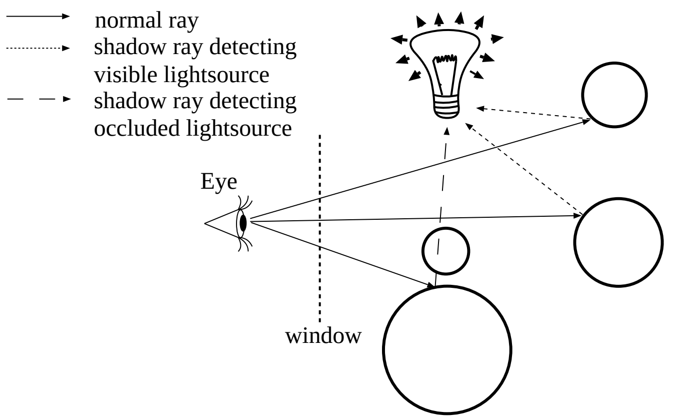

normal ray
shadow ray detecting visible lightsource
shadow ray detecting occluded lightsource
Eye
window

Figure 8.2: Ray-casting

The implementation of the "Trace" function of ray-casting is:

```
Trace(ray)
    (object, $\vec{x}$) = FirstIntersect(ray)
    **if** no intersection **then return** $L_{\text{sky}}$
    color = $L^e(\vec{x}$, -ray.direction ) + DirectLightsource($\vec{x}$, -ray.direction )
    **return** color
**end**
```

In this algorithm L_{sky} is the radiance of background illumination (e.g. sky), "FirstIntersect" is responsible for finding that object which is first intersected by the ray and also the intersection point.

"DirectLightsource", on the other hand, computes an estimate of the single reflection of the light of the light-sources, which happens at point $\vec{x}$ into the given direction. For example, when the scene contains l point light-sources at locations $\vec{y}_1, \dots \vec{y}_l$ with powers $\Phi_1, \dots, \Phi_l$, respectively, and sky-light of radiance L_{sky}, then their reflection at point $\vec{x}$ is:

$$L^{\text{ref}}(\vec{x},\omega) = L_{\text{sky}} \cdot a(\vec{x},\omega) + \sum_{i=1}^{l} \frac{\Phi_l}{4\pi|\vec{y}_i - \vec{x}|^2} \cdot v(\vec{y}_i,\vec{x}) \cdot f_r(\omega_{\vec{y}_i\to\vec{x}},\vec{x},\omega) \cdot \cos\theta_i', \tag{8.2}$$

where θ_i' is the angle between $\omega_{\vec{y}_i\to\vec{x}}$ and the surface normal, and $v(\vec{y}_i,\vec{x})$ indicates the mutual visibility of two points, determined by the **shadow ray**s. In order to handle an area lightsource of emission $L^e(\vec{y},\omega)$, Monte-Carlo integration can be used for equation (7.17), which selects N uniformly distributed $\vec{y}_i$ samples on the lightsource area of size A and applies the following estimate:

$$L^{\text{ref}}(\vec{x},\omega) \approx \frac{A}{N} \cdot \sum_{i=1}^{N} L^e(\vec{y}_i,\omega_{\vec{y}_i\to\vec{x}}) \cdot v(\vec{y}_i,\vec{x}) \cdot f_r(\omega_{\vec{y}_i\to\vec{x}},\vec{x},\omega) \cdot \frac{\cos\theta_i' \cdot \cos\theta_{\vec{y}_i}}{|\vec{x}-\vec{y}_i|^2}. \tag{8.3}$$

8.1.2 Visibility ray-tracing

Visibility ray-tracing is a recursive ray-tracing like algorithm which can follow multiple light bounces only for ideal reflection and refraction (it is of $L[D]S^*E]$ type).

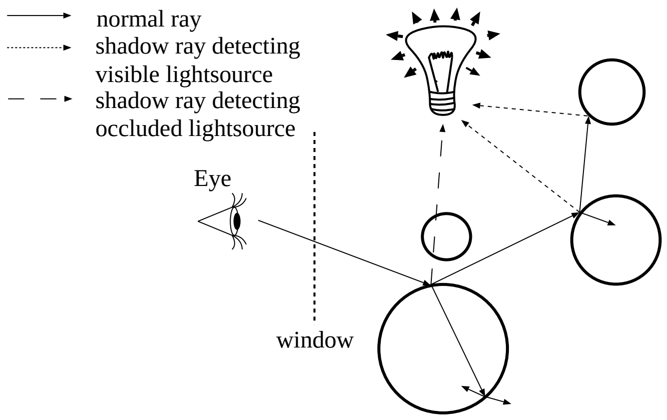

normal ray
shadow ray detecting visible lightsource
shadow ray detecting occluded lightsource
Eye
window

Figure 8.3: Visibility ray-tracing

Formally, it simplifies the rendering equation to the following form:

$$L(\vec{x},\omega) = L^e(\vec{x},\omega) + \int_{\Omega} L_{\text{lightsource}}(h(\vec{x},-\omega'),\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta' \, d\omega' +$$

$$k_r(\omega_r,\vec{x},\omega) \cdot L(h(\vec{x},-\omega_r),\omega_r) + k_t(\omega_t,\vec{x},\omega) \cdot L(h(\vec{x},-\omega_t),\omega_t), \tag{8.4}$$

where ω_r and ω_t are the ideal reflection and refraction directions, and k_r and k_t are the reflection and refraction coefficients.

The implementation of the "Trace" function of the visibility ray-tracing is:

```
Trace(ray)
    (object, $\vec{x}$) = FirstIntersect(ray)
    if no intersection then return $L_{\text{sky}}$
    color = $L^e(\vec{x}$, -ray.direction ) + DirectLightsource($\vec{x}$, -ray.direction )
    if $k_r > 0$ then color += $k_r\cdot$ Trace(reflected ray)
    if $k_t > 0$ then color += $k_t\cdot$ Trace(refracted ray)
    return color
end
```

This subroutine calls itself recursively to find the radiance of the illumination at the reflection and refraction directions. In order to avoid infinite recursion, the algorithm is usually extended to limit the maximum level of recursion.

8.1.3 Distributed ray-tracing

Distributed ray-tracing suggested by Cook [CPC84] is a global illumination algorithm, which can model all the possible paths.

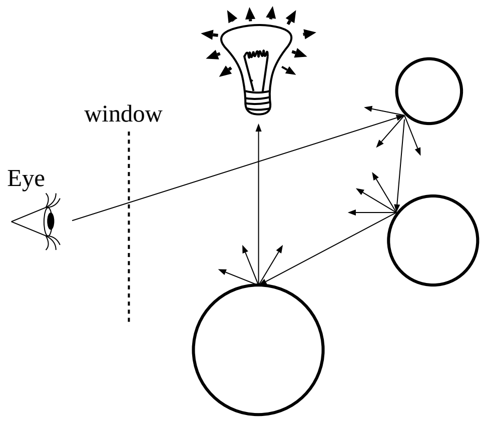


Figure 8.4: Distributed ray-tracing

In this method the ray tracing is not terminated when reaching a surface having neither ideal reflection nor ideal refraction. After a ray has hit a diffuse surface, child rays are generated randomly according to the BRDF characterizing the surface. For the appropriate estimation of the general interreflection, child rays have to be traced and the average of their contributions have to be computed. This approach is based on the recursive formulation of the integrals in the Neumann series (equation (4.10)).

The implementation of the "Trace" function of distributed ray-tracing is:

```
Trace(ray)
    (object, $\vec{x}$) = FirstIntersect(ray)
    **if** no intersection **then return** $L_{\text{sky}}$
    color = $L^e(\vec{x}$, -ray.direction ) + DirectLightsource($\vec{x}$, -ray.direction )
    **for** sample = 1 **to** $N$ **do**
        prob = BRDFSampling(-ray.direction, normal, newray)
        **if** prob $> 0$ **then** color += Trace( newray ) $\cdot$ $w$(newray.direction, normal, -ray.direction) / prob /$N$
    **endfor**
    **return** color
**end**
```

In this program "BRDFSampling" — as defined in section 7.4.2 — finds a new ray to follow, which is then traced recursively. If "BRDFSampling" returns with 0, then the color is set to 0 without recursive calls.

8.1.4 Path-tracing

Another Monte-Carlo approach proposed by Kajiya is **path-tracing** [Kaj86], which is based on the multi-dimensional integral formulation of the terms of the Neumann series (equation (4.7)).

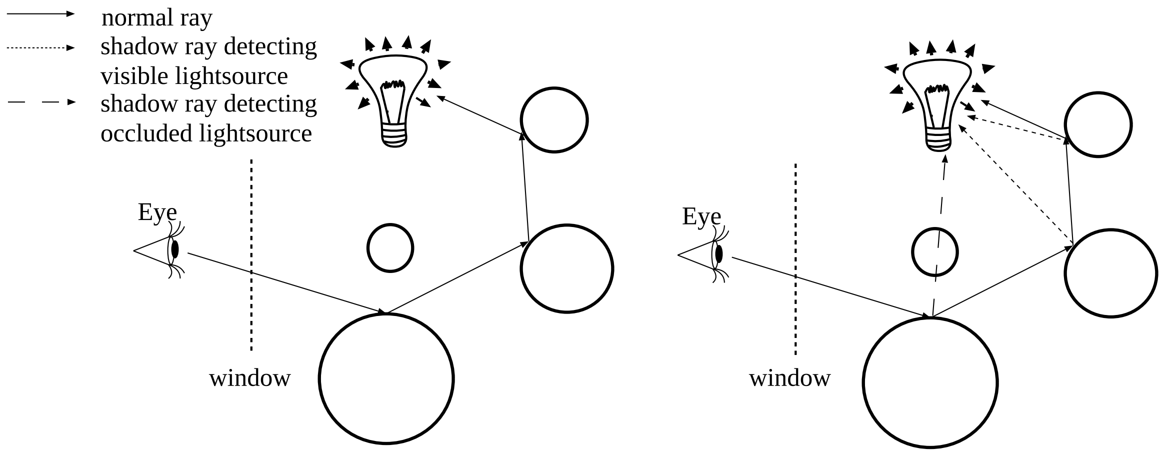

normal ray
shadow ray detecting visible lightsource
shadow ray detecting occluded lightsource
Eye
window
Eye
window

Figure 8.5: Path tracing without (left) and with (right) direct lightsource computation

This method creates a path history for a single particle interacting with the environment until absorption using BRDF sampling and Russian roulette. Rather than spawning new rays at each intersection, it chooses a random direction according to a density t_i which is approximately proportional to w_i. The walk is continued with a probability $a_i = w_i/t_i$ which is equal to the approximation of the albedo (Russian roulette). The measured value of a single path is

$$P = c \cdot (L_1^e + L_2^e \cdot \frac{w_1}{t_1 \cdot a_1} + L_3^e \cdot \frac{w_2}{t_2 \cdot a_2} \cdot \frac{w_1}{t_1 \cdot a_1} + \ldots)$$

where L_i^e is the emission of the point visited at step i of the path and w_i is the transfer density of this point, and c is the scaling factor of the measurement device. Note that if ideal BRDF sampling is used, then w_i is proportional to t_i and both w_i/t_i and a_i are equal to the albedo, which results in the following estimate:

$$P = c \cdot (L_1^e + L_2^e + L_3^e + \ldots).$$

This estimate has very high variation if the lightsources are small. This problem can be solved if lightsource sampling is combined with the gathering walk, which means that at each visited point the effects of the lightsources are estimated.

The implementation of the "Trace" function of path-tracing is:

```
Trace(ray)
    (object, $\vec{x}$) = FirstIntersect(ray)
    **if** no intersection **then return** $L_{\text{sky}}$
    color = $L^e(\vec{x}$, -ray.direction )+ DirectLightsource($\vec{x}$, -ray.direction )
    prob = BRDFSampling(-ray.direction, normal, newray)
    **if** prob = 0 **then return** color
    color += Trace( newray ) $\cdot$ $w$(newray.direction, normal, -ray.direction) / prob
    **return** color
**end**
```

In this program "BRDFSampling" finds a new direction or if it returns 0, then it has decided that the walk has to be stopped because of Russian-roulette. Note that this algorithm generates all but the last directions of the path by BRDF sampling and the last is obtained by lightsource sampling. Thus if the surface at the light reflection is shiny (close to ideal mirror or ideal refractor), then the quality of importance sampling can be quite bad. Since almost ideal surfaces close to the lightsources are responsible for **caustics**, path tracing — as other gathering algorithms — is poor in rendering caustics effects.

8.2 Shooting-type walks methods

Shooting walks are based on the Monte-Carlo solution of the potential equation. They start at the eye, go through the scene and try to find the eye.

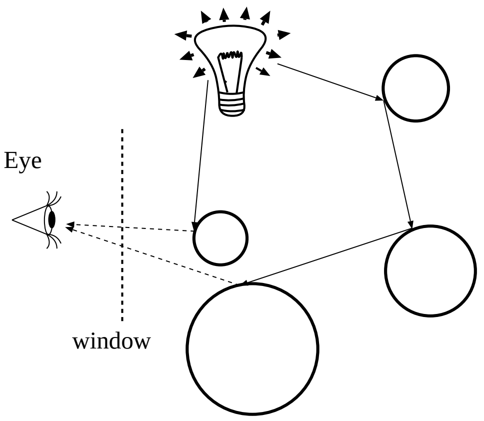


Figure 8.6: Shooting-type walks

The general structure of shooting algorithms is as follows:

```
Clear Image
**for** $i$ = 1 **to** $N$ **do**
    ray = Sample randomly from a lightsource with selection probability $p^e$
    power = $L^e \cdot \cos\theta / p^e / N$
    Shoot( ray, power )
**endfor**
```

In different shooting algorithms the “Shoot” function is implemented differently. This function is responsible for determining the power carried to the eye by the complete path and also for the identification of the pixel through which the path arrives at the eye.

8.2.1 Photon tracing

Photon tracing (forward ray-tracing) is the inverse of visibility ray-tracing and uses similar simplifying assumptions.

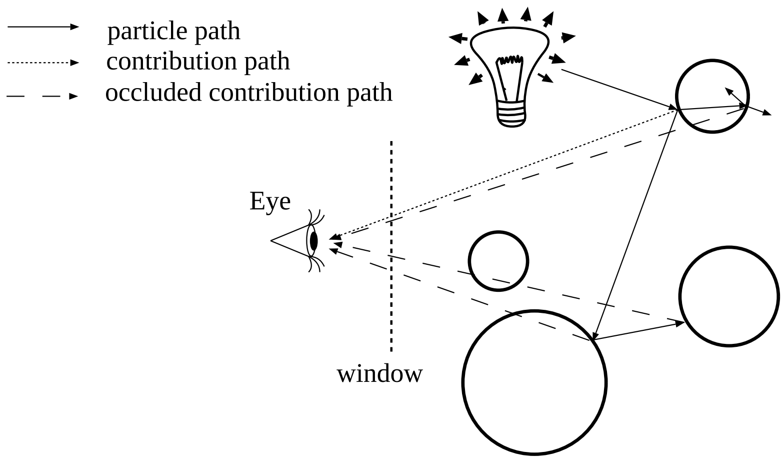


Figure 8.7: Photon tracing

It also stops tracing when hitting a surface that does not have coherent reflection or refraction. In photon tracing the rays are emitted from the lightsources, and at each hit it is examined whether the surface has ideal reflection, refraction and incoherent reflection or refraction. In the directions of ideal reflection or refraction, the tracing is continued by starting new child rays.

The implementation of its Shoot function is:

```
Shoot(ray, power)
    (object, $\vec{x}$) = FirstIntersect(ray)
    **if** no intersection **then return**
    **if** $\vec{x}$ is visible from pixel $p$ **then**
        color[$p$] += power $\cdot w$(ray.direction, $\vec{x}$, eye direction ) $\cdot\ g(\vec{x})$
    **endif**
    **if** $k_r > 0$ **then** Shoot( reflected ray, $k_r \cdot$ power )
    **if** $k_t > 0$ **then** Shoot( refracted ray, $k_t \cdot$ power )
    **return**
**end**
```

The “eye direction” is a vector pointing from $\vec{x}$ to the eye position. The algorithm is capable of handling LS^*DE paths.

8.2.2 Light-tracing

In **light-tracing** [DLW93] photons perform random walk through the scene starting at the lightsources. Whenever a surface is hit, a ray is traced from the intersection point to the eye and the contribution is added to the selected pixel (if any).

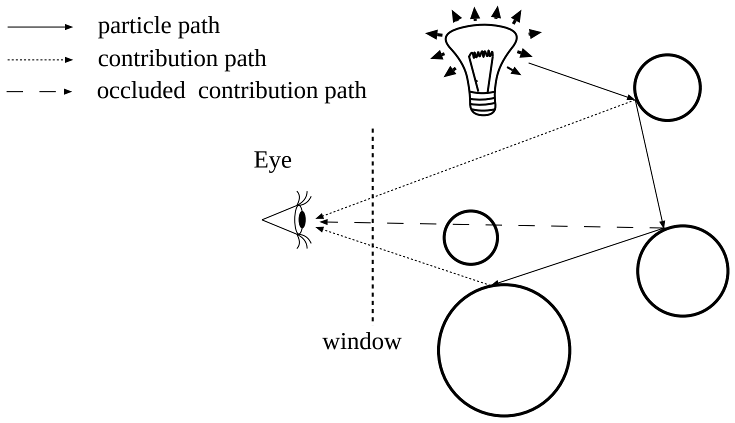


Figure 8.8: Light tracing

Light tracing is the direct implementation of the Monte-Carlo quadrature of the multi-dimensional formulation of the potential equation. When the next direction is determined, BRDF based importance sampling can be applied and combined with Russian-roulette. It chooses a random direction according to a density t_i which is approximately proportional to w_i (importance sampling). The walk is continued with a probability a_i equal to the approximation of the albedo (Russian roulette). The measured value of a single step of the path is

$$P = \frac{L^e \cos\theta}{N \cdot p^e} \cdot \frac{w_1}{t_1 \cdot a_1} \cdot \frac{w_2}{t_2 \cdot a_2} \cdot \ldots \cdot w(eye) \cdot g,$$

if this point is visible at the pixel and zero otherwise. Here L^e is the emission of the starting point, θ is the angle between the surface normal of the lightsource and the first direction, p^e is the probability of selecting this lightsource point and starting direction, $w(eye)$ is the cosine weighted BRDF at the given point from the last direction to the eye, and g is the surface dependent camera parameter. Note that if ideal BRDF sampling is used, i.e. w_i is proportional to t_i and both w_i/t_i and a_i are equal to the albedo, and ideal lightsource sampling is used, i.e. p^e is proportional to $L^e \cos\theta$, thus $L^e \cos\theta / N \cdot p^e = \Phi/N$, the following estimate can be obtained:

$$P = \frac{\Phi}{N} \cdot w(eye) \cdot g.$$

This estimate has high variation if the camera is often hidden since if the point is not visible from the camera, the contribution is zero.

The implementation of the “Shoot” function of light tracing is:

```
Shoot(ray, power)
    (object, $\vec{x}$) = FirstIntersect(ray)
    **if** no intersection **then return**
    **if** $\vec{x}$ is visible from pixel $p$ **then**
        color[$p$] += power $\cdot w$(ray.direction, $\vec{x}$, eye direction ) $\cdot\, g(\vec{x})$
    **endif**
    prob = BRDFSampling(-ray.direction, normal, newray )
    **if** prob = 0 **then return**
    newpower = power * $w$( -ray.direction, normal, newray.direction ) / prob
    Shoot( newray, newpower )
    **return**
**end**
```

This algorithm also applies BRDF sampling for all but the last steps. The direction of the last visibility ray might be far from the direction preferred by the BRDF. This degrades the performance of importance sampling if the visible surface is very shiny. Thus visible mirrors or refractors (glass) pose difficulties to shooting algorithms.

8.2.3 Random walks for the radiosity setting

Expansion expands the solution into a discrete Neumann series

$$\mathbf{L} = \mathbf{L}^e + \mathbf{R} \cdot \mathbf{L}^e + \mathbf{R}^2 \cdot \mathbf{L}^e + \mathbf{R}^3 \cdot \mathbf{L}^e + \ldots \tag{8.5}$$

Let us again examine the $\mathbf{R}^2 \cdot \mathbf{L}^e$ term. Using the definition of the matrix $\mathbf{R}$, this can also be expressed as a multi-dimensional integral:

$$(\mathbf{R}^2 \cdot \mathbf{L}^e)|_i = \sum_{j=1}^{n} \sum_{k=1}^{n} \mathbf{R}_{ij} \cdot \mathbf{R}_{jk} \cdot \mathbf{L}^e_k =$$

$$\int_S \int_\Omega \int_S \int_\Omega \tilde{b}_i(\vec{x}_1) \cdot w_1(i) \cdot \sum_{j=1}^{n} b_j(h(\vec{x}_1, -\omega'_1)) \cdot \tilde{b}_j(\vec{x}_2) \cdot w_2(j) \cdot \sum_{k=1}^{n} b_k(h(\vec{x}_2, -\omega'_2)) \cdot \mathbf{L}^e_k \, d\omega'_2 d\vec{x}_2 d\omega'_1 d\vec{x}_1,$$

where

$$\begin{aligned} w_1(i) &= f_i \cdot \cos\theta'_1, \\ w_2(j) &= f_j \cdot \cos\theta'_2. \end{aligned} \tag{8.6}$$

Considering the integrand, $\vec{x}_1$ should be in patch i for $\tilde{b}_i$ to be non zero. Then, only a single b_j will give non-zero value for the $\vec{y}_1 = h(\vec{x}_1, -\omega'_1)$ point. To select this, a ray has to be traced from $\vec{x}_1$ in direction $-\omega'_1$ and the visible patch should be identified. Following this, another point on the identified patch i should be selected, which is denoted by $\vec{x}_2$, and a ray is traced in direction $-\omega'_2$ to obtain an index k of a patch whose emission should be propagated back on the walk. During propagation, the emission is multiplied by the BRDFs (f_i, f_j) and the cosine ($\cos\theta'_2, \cos\theta'_1$) factors of the visited patches (figure 8.9).

Note that this is basically the same walking scheme, as used to solve the original integral equation. The fundamental difference is that when a patch is hit by the ray, the walk is not continued from the found point but from another point of the patch.

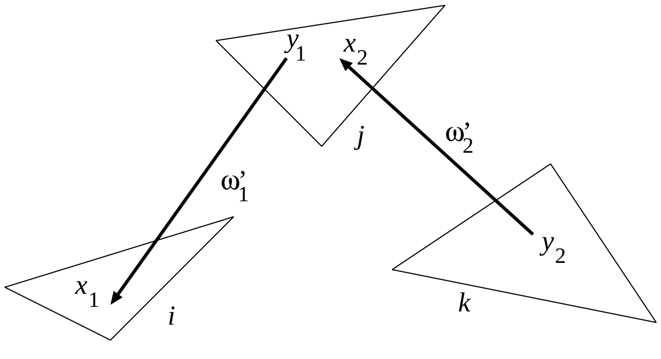


Figure 8.9: Random walk solution of the projected rendering equation

The power equation can be treated similarly. Again, let us examine the two-bounce case

$$(\mathbf{H}^2 \cdot \mathbf{P}^e)|_i = \sum_{j=1}^{n} \sum_{k=1}^{n} \mathbf{R}_{ji} \cdot \frac{f_i}{f_j} \cdot \mathbf{R}_{kj} \cdot \frac{f_j}{f_k} \cdot \mathbf{P}^e_k =$$

$$\int_S \int_\Omega \int_S \int_\Omega \mathbf{P}^e_k \cdot \tilde{b}_k(\vec{y}_1) \cdot w_1(k) \cdot \sum_{j=1}^{n} b_j(h(\vec{y}_1, \omega_1)) \cdot \tilde{b}_j(\vec{y}_2) \cdot w_2(j) \cdot \sum_{k=1}^{n} b_i(h(\vec{y}_2, \omega_2)) \cdot w_3(i) \, d\omega_2 d\vec{y}_2 d\omega_1 d\vec{y}_1,$$

where

$$\begin{aligned} w_1(k) &= \cos\theta_1, \\ w_2(j) &= f_j \cdot \cos\theta_2, \\ w_3(i) &= f_i. \end{aligned} \tag{8.7}$$

It means that the integrand in a single point can be obtained by selecting a point $\vec{y}_1$ on patch k, then tracing a ray in direction ω_1. Having identified the intersected patch j a new point $\vec{y}_2$ is selected on this patch and the ray-tracing is continued at direction ω_2. The patch which is hit by this ray receives the power of patch k attenuated by the BRDFs and the cosine factors of the steps.

Thus, the projected rendering equation can also be solved by random walks [Shi91b, Sbe96]. The basic difference is that when a patch is hit by a ray, then instead of initiating the next ray from this point, another independent point is selected on the same patch.

Considering the concept of importance sampling and Russian roulette, many different strategies can be elaborated by appropriately defining the P, $\tilde{W}$ and $\tilde{L}^e$ functions (recall that according to equation (7.33) the requirement of an unbiased estimate is $P(\mathbf{z}_1, \ldots, \mathbf{z}_n) \cdot \tilde{W} \cdot \tilde{L}^e = W \cdot L^e$). For example, let us use the following simulation [Shi91b, Sbe96] to obtain a radiance estimate of patch i_1:

First a ray is found that starts on this patch. The starting point $\vec{x}_1$ is sampled from a uniform distribution, while the direction ω_1' is sampled from a cosine distribution, thus the probability density is $1/A_{i_1} \cdot \cos\theta_1'/\pi$. This ray is traced and the next patch is identified. Let it be patch i_2. At patch i_2 it is decided whether or not the walk should be stopped with probability of the albedo of the patch. Note that for diffuse surfaces the albedo is $a = f \cdot \pi$. If the walk has to be continued, then a new starting point $\vec{x}_2$ is found on patch i_2, and the same procedure is repeated recursively.

With this strategy, the probability density of completing an n step walk is

$$p(\vec{x}_1, \omega_1', \vec{x}_2, \omega_2', \ldots \vec{x}_{n-1}, \omega_{n-1}') = \frac{1}{A_{i_1}} \cdot \frac{\cos\theta_1'}{\pi} \cdot \frac{a_{i_2}}{A_{i_2}} \cdot \frac{\cos\theta_2'}{\pi} \cdots \frac{a_{i_{n-1}}}{A_{i_{n-1}}} \cdot \frac{\cos\theta_{n-1}'}{\pi} \cdot (1 - a_{i_n}) =$$

$$\frac{f_{i_1}}{A_{i_1}} \cdot \cos\theta_1' \cdot \frac{f_{i_2}}{A_{i_2}} \cdot \cos\theta_2' \cdots \frac{f_{i_{n-1}}}{A_{i_{n-1}}} \cdot \cos\theta_{n-1}' \cdot \frac{1 - a_{i_n}}{a_{i_1}} = W \cdot \frac{1 - a_{i_n}}{a_{i_1}}. \tag{8.8}$$

Thus the required weight $\tilde{W}$ of the walk is

$$\tilde{W} = \frac{a_{i_1}}{1 - a_{i_n}}. \tag{8.9}$$

Thus if the patch on which the walk is terminated is a source having emission $\mathbf{L}_n^e$, then the estimator of the radiance of patch i is

$$\mathbf{L}_n^e \cdot \frac{a_{i_1}}{1 - a_{i_n}}.$$

Other gathering or shooting estimators have been proposed and their variances have been estimated in [Shi91b, Sbe96].

8.3 Bi-directional random walk algorithms

Bi-directional algorithms is based on the combination of shooting and gathering walks, thus it can combine the advantages of both techniques. Namely, it can effectively handle small lightsources and small aperture cameras and can render caustics and ideally refracting or reflecting visible objects.

8.3.1 Bi-directional path-tracing

Bi-directional path-tracing [LW93, VG95] initiates paths at the same time from a selected lightsource and from the viewpoint. After some steps, either a single deterministic shadow ray is used to connect the two types of walks [VG95], or all points of the gathering walk are connected to all points of the shooting walk using deterministic rays [LW93]. If the deterministic shadow ray detects that the two points are occluded from each other, then the contribution of this path is zero.

Note that gathering and shooting walks use different integration variables, namely a gathering walk is specified by a point on the pixel area and a sequence of incoming directions, while a shooting walk is defined by a point on the lightsource and a sequence of the outgoing directions. Thus when the two walks are connected, appropriate transformations should take place.

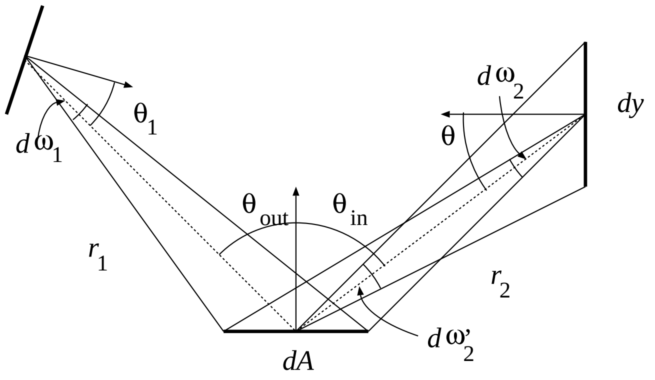


Figure 8.10: Correspondence between the solid angles of incoming and outgoing directions

Let us first consider a walk of a single bounce (figure 8.10). According to the definition of the solid angle, we obtain

$$\frac{d\omega_1'}{d\omega_2} = \frac{dA \cdot \cos\theta_{\rm out}/r_1^2}{dA \cdot \cos\theta_{\rm in}/r_2^2} = \frac{r_2^2}{r_1^2} \cdot \frac{\cos\theta_{\rm out}}{\cos\theta_{\rm in}}, \tag{8.10}$$

and for the substitution of the surface integral on the lightsource

$$d\omega_2' = \frac{d\vec{y} \cdot \cos\theta}{r_2^2}. \tag{8.11}$$

Thus the transformation rule is

$$\cos\theta_1' \cdot \cos\theta_{\rm in}\; d\omega_1' d\omega_2' = \frac{\cos\theta_1' \cdot \cos\theta_{\rm out}}{r_1^2} \cdot \cos\theta\; d\omega_2 d\vec{y},$$

which means that when converting a shooting type walk to a gathering type walk, then the radiance should be multiplied by

$$\frac{\cos\theta_1' \cdot \cos\theta_{\rm out}}{r_1^2}.$$

When the shooting walk consists of more than 1 steps, then formula (8.10) should be applied to each of them, but formula (8.11) only to the last step. This conversion replaces the incoming directions by the outgoing directions and the subsequent steps compensate r_{k+1}^2/r_k^2 scaling. Finally, we end up with a formula which is similar to the 1-step case:

$$\cos\theta_k' \cdot \cos\theta_{k+1}' \cdot \ldots \cos\theta_n'\; d\omega_k' \ldots d\omega_n' = \frac{\cos\theta_k' \cdot \cos\theta_{n-k+1}}{r_k^2} \cdot \cos\theta_{n-k} \cdot \ldots \cos\theta_1\; d\omega_{n-k} \ldots d\omega_1 d\vec{y}.$$

Figure 8.11 shows an example when $k = 2$ and $n = 4$. This formula means that we can use the rules of sections 4.2.1 and 4.2.2 to generate the shooting and gathering walks — gathering walks use the cosine of the incoming

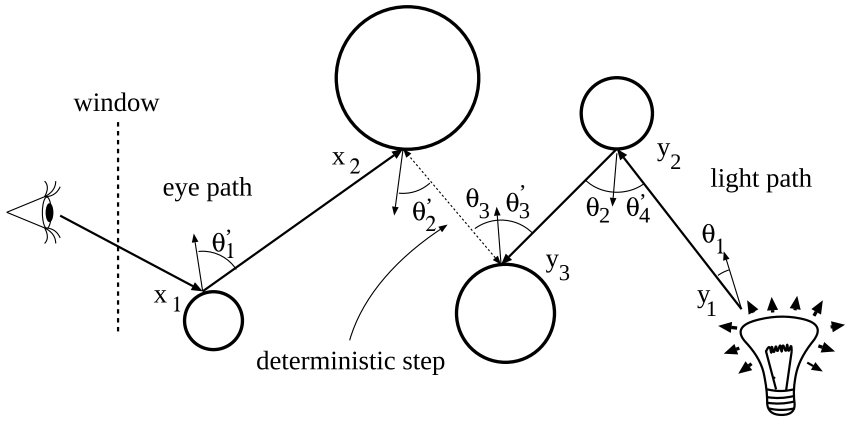


Figure 8.11: Bi-directional path tracing with a single deterministic step

angle, while shooting walks use the cosine of the outgoing angle — and the transformation of the combined walk to a single gathering walk requires a multiplication by

$$\frac{\cos\theta'_k \cdot \cos\theta_{n-k+1}}{r_k^2}.$$

Formally, if the endpoints of the shooting and gathering walks are visible from each-other, then the measured value of a single path is

$$P = \frac{L^e \cos\theta}{N \cdot p^e} \cdot \frac{w_1^s}{t_1^s \cdot a_1^s} \cdot \frac{w_2^s}{t_2^s \cdot a_2^s} \cdot \ldots \cdot f_r^s \cdot \frac{\cos\theta^s \cdot \cos\theta'^g}{r^2} \cdot f_r^g \cdot \ldots \cdot \frac{w_2^g}{t_2^g \cdot a_2^g} \cdot \frac{w_1^g}{t_1^g \cdot a_1^g} \cdot c$$

where superscripts s and g stand for shooting and gathering steps, respectively. Note that if ideal BRDF sampling is used, then the estimate is:

$$P = \frac{\Phi}{N} \cdot f_r^s \cdot \frac{\cos\theta^s \cdot \cos\theta'^g}{r^2} f_r^g \cdot c.$$

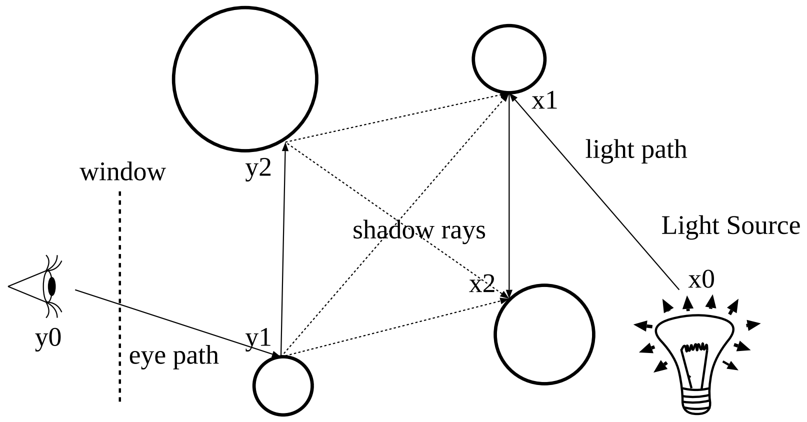


Figure 8.12: Bi-directional path tracing with multiple deterministic steps

In Lafortune's version of the bi-directional path tracing [LW93] not only the endpoints of the shooting and gathering walks are connected, but all intersection points are linked by shadow rays.

Note that in bi-directional path-tracing a path of given length n can be generated by many ways, for instance, by a gathering walk of length i combined with a shooting walk of length $n - i$, where $i = 0, \ldots, n$. This creates a danger of accounting a walk more than one time. To eliminate this danger, the result can be estimated by a weighted sum of the different walks as suggested by the concept of multiple importance sampling. Other heuristic weight factors can also provide good results [LW93, Vea97]. For example, when a gathering walk is computed, at each step the radiance coming from the continuation of the walk is weighted by W and the radiance coming from deterministic connections to shooting walks is weighted by $1 - W$. Since for shiny surfaces the continuation of the walk by BRDF sampling is supposed to be better, W may express how shiny the surface is.

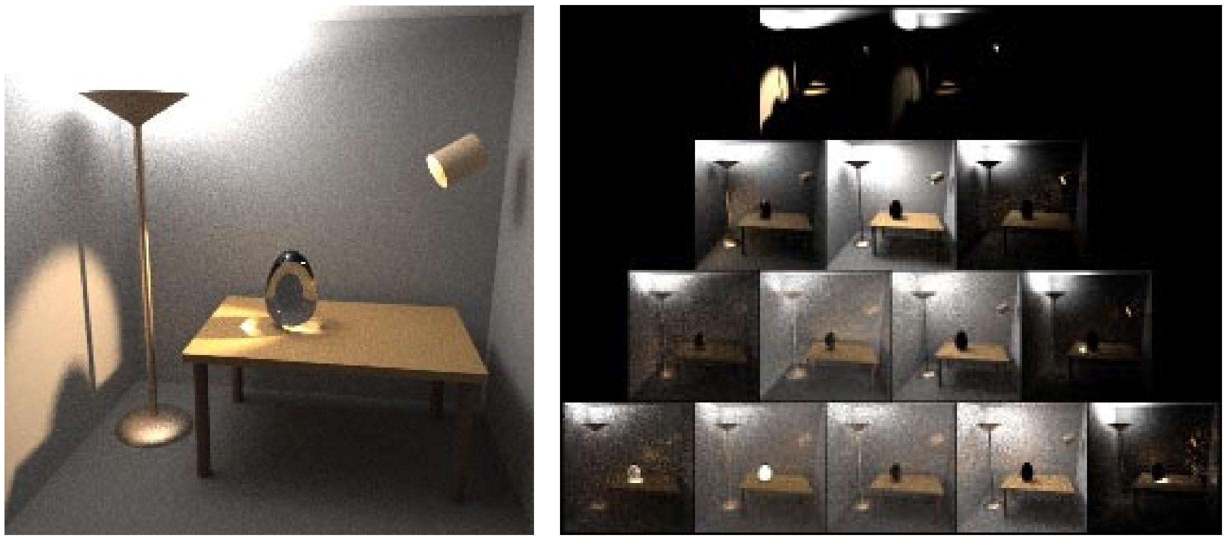


Figure 8.13: Bi-directional path tracing: the final image (left) is the composition of walks consisting of different number of gathering and shooting steps (right) [VG95]

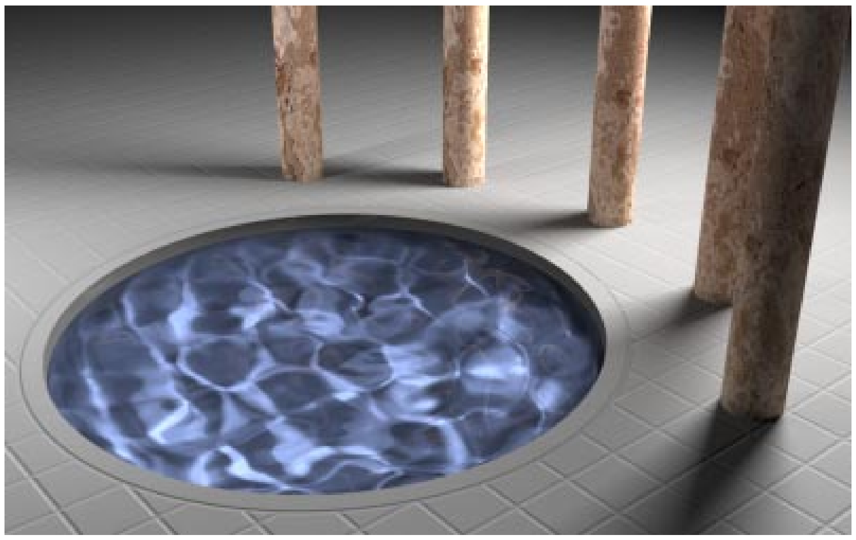


Figure 8.14: Pool rendered with Metropolis light transport (Eric Veach [VG97])

8.3.2 Metropolis light transport

Random walk methods usually generate the ray-paths independently. Thus when a difficult path is found, it is thrown away right after its application. Metropolis method, on the other hand, generates samples by perturbing the previous path, thus they are expected to be better for difficult lighting conditions.

Recall that Metropolis method [MRR$^+$53] offers samples with a probability density that is proportional to a given "importance function" in an asymptotic case. Let this importance function $\mathcal{I}$ be the luminance of the light carried by a ray-path to the eye through any pixel. This selection is justified by the fact that the eye is particularly sensitive to luminance variations and different pixels have equal importance in the image.

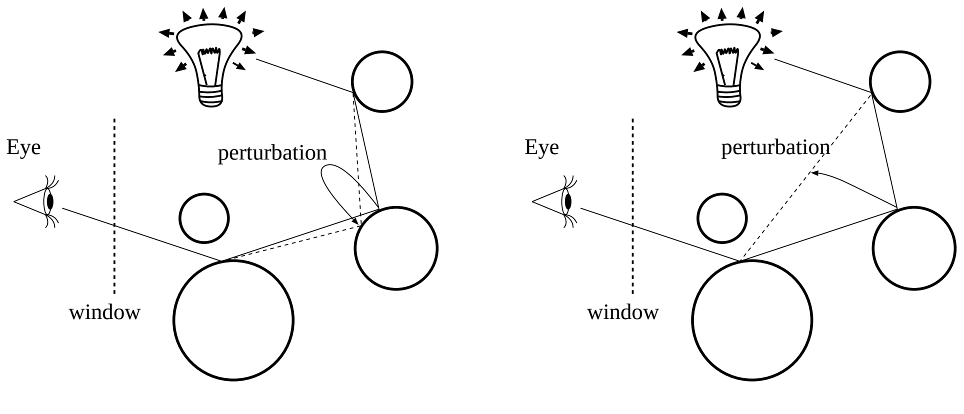


Figure 8.15: Generating walks by mutations in Metropolis light transport

In order to generate samples according to $\mathcal{I}(\mathbf{z})/b$, a Markov process is constructed whose stationary distribution is just this (here $\mathbf{z}$ denotes a ray-path). Scalar b means the integral of the importance function on the whole domain, which can be estimated in a preprocessing phase using a normal random walk algorithm. The Metropolis algorithm generates a sequence of ray-paths starting from an initial path. Veach and Guibas proposed bi-directional path tracing [VG97] to find an initial path, although any random walk algorithm can be used for this. Generating a new path $\mathbf{z}_{i+1}$ after path $\mathbf{z}_i$ consists of two steps. First the **tentative transition function** $T(\mathbf{z}_i \to \mathbf{z}_t)$ produces a **tentative path** $\mathbf{z}_t$ by mutating path $\mathbf{z}_i$ a little, i.e. by changing directions and adding or deleting steps. Then the tentative path is either accepted or rejected using an **acceptance probability**

$$a(\mathbf{z}_i \to \mathbf{z}_t) = \min\left\{\frac{\mathcal{I}(\mathbf{z}_t)\cdot T(\mathbf{z}_t \to \mathbf{z}_i)}{\mathcal{I}(\mathbf{z}_i)\cdot T(\mathbf{z}_i \to \mathbf{z}_t)},\, 1\right\}$$

that expresses the increase of the importance. The definition of mutations is almost arbitrary if they make the Markov process ergodic. Ergodic processes have stationary distribution, and this distribution is independent of the starting state of the process. Practically it requires that any path of positive power could be generated from any other path after a certain number of perturbations. Thus mutations should modify all features, including directions (or visited points), starting point, length, etc. Furthermore, in order to avoid situations when the process is stuck into a region which is surrounded by regions of zero importance, the tentative transition should take large enough steps to jump over these zero importance regions. Veach [VG97], for example, proposed the generation of a completely new path when the contribution of the tentative path is zero.

Summarizing, the **Metropolis light-transport** algorithm is:

```
Generate an initial ray-path $\mathbf{z}_1$ using random walk, e.g. bi-directional path tracing
for $i = 1$ to $N$ do
    Based on the actual ray-path, find another, tentative path $\mathbf{z}_t$ mutating $\mathbf{z}_i$ with $T(\mathbf{z}_i \to \mathbf{z}_t)$
    if $\mathcal{I}(\mathbf{z}_t) = 0$ then Generate a completely new path $\mathbf{z}_{i+1}$ from scratch using random walk
    else
        $a(\mathbf{z}_i \to \mathbf{z}_t) = (\mathcal{I}(\mathbf{z}_t)\cdot T(\mathbf{z}_t \to \mathbf{z}_i))/(\mathcal{I}(\mathbf{z}_i)\cdot T(\mathbf{z}_i \to \mathbf{z}_t))$
        Generate uniformly distributed random number $r$ in $[0,1]$        // accept with "probability" $a(\mathbf{z}_i \to \mathbf{z}_t)$
        if $r < a(\mathbf{z}_i \to \mathbf{z}_t)$ then $\mathbf{z}_{i+1} = \mathbf{z}_t$ else $\mathbf{z}_{i+1} = \mathbf{z}_i$
    endif
    Compute the contribution of the ray-path $\mathbf{z}_{i+1}$ to the affected pixel
    Multiply this contribution by $b/(\mathcal{I}(\mathbf{z}_{i+1})\cdot N)$ and accumulate to the pixel
endfor
```

8.3.3 Photon-map

Bi-directional path tracing connects a single gathering walk to a single shooting walk. However, if the effects of all shooting walks could be stored, then when a new gathering walk is computed, it could be connected to all of the shooting walks simultaneously, which can significantly increase the number of samples in the integral quadrature. This is exactly what Jensen [JC95, Jen96, JC98] proposed, also giving the definition of a data structure, called the **photon-map** which can efficiently store the effects of many shooting walks.

A photon map is a collection of photon hits at the end of the paths generated in the shooting phase of the algorithm. The photon-map is organized in a **kd-tree** to support efficient retrieval. A photon hit is stored with the power of the photon on different wavelengths, position, direction of arrival and with the surface normal.

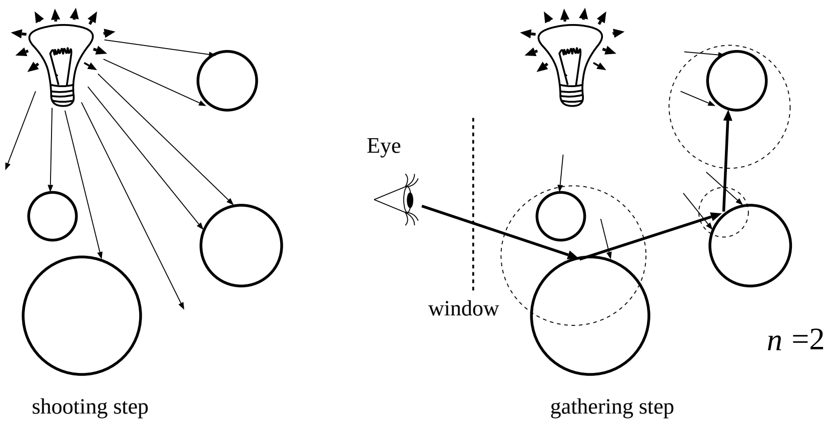


Figure 8.16: Rendering with photon-map

The gathering phase is based on the following approximation of the transport operator:

$$L(\vec{x},\omega') = \int_\Omega L(h(\vec{x},-\omega'),\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta' \, d\omega' = \int_\Omega \frac{d\Phi(\omega')}{dA\cos\theta' d\omega'} \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta' \, d\omega' \approx$$

$$\sum_{i=1}^{n} \frac{\Delta\Phi(\omega_i')}{\Delta A} \cdot f_r(\omega_i',\vec{x},\omega), \tag{8.12}$$

where $\Delta\Phi(\omega_i')$ is the power of a photon landing at the surface ΔA from direction ω_i'.

The $\Delta\Phi$ and ΔA quantities are approximated from the photons in the neighborhood of $\vec{x}$ in the following way. A sphere centered around $\vec{x}$ is extended until it contains n photons. If at this point the radius of the sphere is r, then the intersected surface area is $\Delta A = \pi r^2$ (figure 8.17).

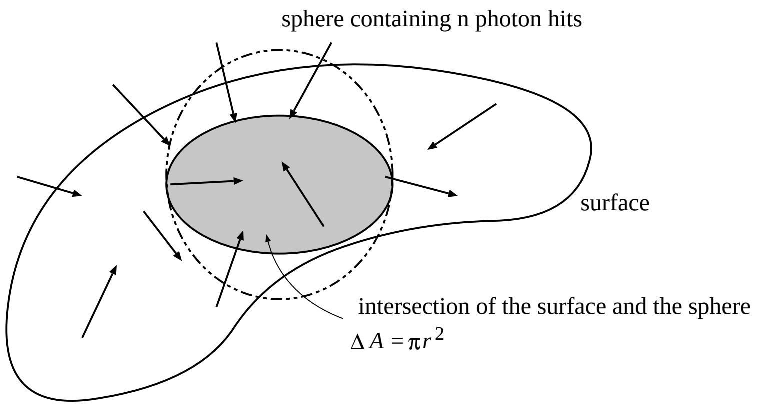

sphere containing n photon hits
surface
intersection of the surface and the sphere
$\Delta A = \pi r^2$

Figure 8.17: Retrieving data from the photon-map

Figure 8.18: Application of photon maps (Mental Images)

8.3.4 Instant radiosity

Instant radiosity [Kel97] elegantly subdivides the shooting walks into a view-independent walk and to the last bounce which reflects the contribution to the eye. The view-independent walks are calculated in the first phase of the algorithm and each walk results in a point-lightsource that represents the power at the end of the walk. The view-independent walk is quite similar to the light-tracing algorithm, but the new directions are sampled from the Halton sequence instead of a random distribution. In the radiosity setting the reflection at the end of the walks is also diffuse, thus it can be stored as a point-lightsource of homogeneous directional radiation.

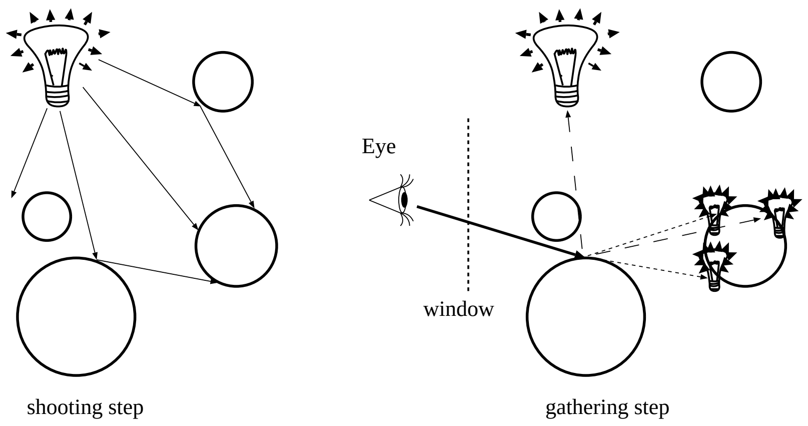


Figure 8.19: Instant radiosity

In the second phase of the algorithm the power of the diffuse lightsources are reflected towards the eye. Since this is a simple, local illumination problem, the algorithm can take advantage of the rendering hardware of advanced workstations which can render the effect of these lightsources on the scene and also to compute shadows. If the number of lightsources is more than what can be handled by the hardware, the computation is broken into different phases. Each phase uses just a few of the lightsources and the final image is obtained as the average of the estimates of the phases, which is computed by the hardware accumulation buffer.

Instant radiosity is quite similar to photon-map based techniques. However, instead of using ray-tracing for final gather, the photons in the photon map are used as lightsources and fast and hardware supported visibility and shadow algorithms are applied. The other fundamental difference is that instant radiosity allows just a relatively low number of photons which therefore should be very well distributed. The optimal distribution is provided by quasi-Monte Carlo light walks.

8.4 Global methods

Algorithms discussed so far use recursive ray-tracing to transfer the light in the scene. Ray-tracing is rather time consuming and is unable to exploit the coherence of the radiance function. Therefore, so called **global method**s transfer the light globally and not just for a single point of the scene.

8.4.1 Multi-path method using global random lines

Multi-path methods represent a bridge between random walk and iteration. They are essentially random walk methods, but in their single step many random walks are advanced simultaneously.

Sbert [Sbe96, SPNP96, SMP98] proposed a complete family of multi-path methods that are based on random **global lines**, which was the basic "engine" to advance the walks. A single global line transfers the reflected power of all those patches that are intersected by this line to the direction of this line. The global line also transfers a portion of the emission of the intersected patches. Thus a line initiates those walks that would start in a patch intersected by this line, and continues those previous walks which carried some power onto the intersected patches.

8.4.2 Global ray-bundle tracing

Realizing that an accurate solution requires great many samples, **global ray-bundle tracing** [SKFP98b, SKP98, SK98a] uses a bundle of very many (e.g. 1 million or even infinite) global parallel rays, which can be traced simultaneously using image coherence techniques. In order to represent the radiance that is transferred by a ray, finite-element techniques are applied that approximate the positional (but not the directional) dependence of the radiance by piece-wise continuous or piece-wise linear functions [SKFP98a].

$$L(\vec{x},\omega) \approx \sum_{j=1}^{n} b_j(\vec{x}) \cdot L_j(\omega) = \mathbf{b}^T \cdot \mathbf{L}(\omega). \tag{8.13}$$

Note that this is a mixed finite-element and continuous method, since the positional dependence of the radiance is approximated by finite-elements, while the directional dependence is not.

Substituting this into the rendering equation and projecting that into an adjoint base we obtain

$$\mathbf{L}(\omega) = \mathbf{L}^e(\omega) + \mathcal{T}_F \mathbf{L}(\omega), \tag{8.14}$$

where $\mathcal{T}_F$ is a composition of the original transport operator and its projection to the adjoint base

$$\mathcal{T}_F \mathbf{L}(\omega)|_i = \frac{1}{A_i} \cdot \int_{\Omega} \int_{A_i} L(h(\vec{x}, -\omega'), \omega') \cdot \cos\theta' \cdot \tilde{f}_i(\omega', \omega) \, d\vec{x} \, d\omega'. \tag{8.15}$$

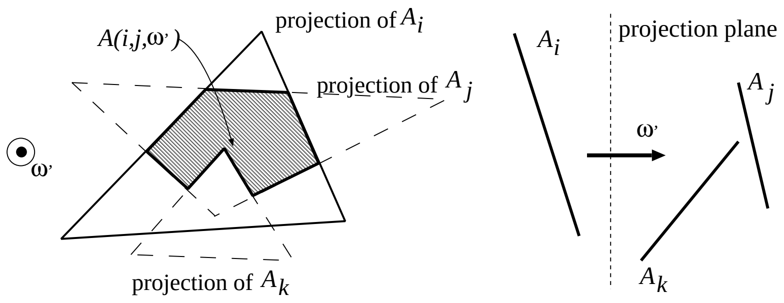


Figure 8.20: Interpretation of $A(i,j,\omega')$

Taking into account that the integrand of the inner surface integral is piece-wise constant, it can also be presented in closed form:

$$\int_{A_i} L(h(\vec{x}, -\omega'), \omega') \cdot \cos\theta' \cdot \tilde{f}_i(\omega', \omega) \, d\vec{x} = \sum_{j=1}^{n} \tilde{f}_i(\omega', \omega) \cdot A(i,j,\omega') \cdot L_j(\omega'), \tag{8.16}$$

where $A(i,j,\omega')$ expresses the projected area of patch j that is visible from patch i in direction ω'. In the unoccluded case this is the intersection of the projections of patch i and patch j onto a plane perpendicular to ω'. If occlusion occurs, the projected areas of other patches that are in between patch i and patch j should be subtracted as shown in figure 8.20.

This projected area can be efficiently calculated simultaneously for all patch pairs using global discrete or continuous visibility algorithms [SK98a] and also exploiting the hardware z-buffer [SKP98]. These algorithms can also have random nature, that is, they can result in $A(i,j,\omega')\cdot L_j(\omega')$ just as an the expected value [SKP99, SK98b].

Using equation (8.16) the rendering equation can be obtained as:

$$\mathbf{L}(\omega) = \mathbf{L}^e(\omega) + \int_{\Omega} \mathbf{F}(\omega',\omega)\cdot\mathbf{A}(\omega')\cdot\mathbf{L}(\omega')\,d\omega', \tag{8.17}$$

where $\mathbf{L}(\omega)$ is the vector of radiance values, $\mathbf{F}(\omega',\omega)$ is a diagonal matrix of BRDFs, and *geometry matrix* $\mathbf{A}$ contains the relative visible areas: $\mathbf{A}(\omega')|_{ij} = A(i,j,\omega')/A_i$.

Note that equation (8.17) is highly intuitive as well. The radiance of a patch is the sum of the emission and the reflection of all incoming radiance. The role of the patch-direction-patch "form-factors" is played by $A(i,j,\omega')/A_i$.

This is also an integral equation but unlike the original rendering equation it provides the radiance of not only a single point but for all points at once. This integral equation is solved by random or quasi-random shooting type walks.

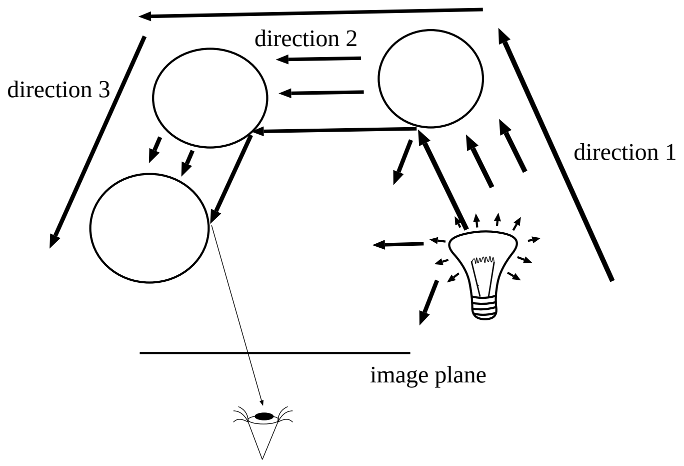


Figure 8.21: A path of ray-bundles

A single walk starts by selecting a direction either randomly or quasi-randomly, and the emission of all patches of the scene is transferred into this direction (figure 8.21). Then a new direction is found, and the emission is transferred again and the incoming radiance generated by the previous transfer is reflected from all patches into this new direction. The algorithm keeps doing this for a few times depending on how many bounces should be considered, then the emission is sent and the incoming radiance caused by the last transfer is reflected towards the eye, resulting an image estimate associated with the walk. The final image can be obtained by averaging the image estimates of different walks.

8.4.3 Preprocessing the point lightsources

Global radiosity methods are efficient for large area lightsources but loses their advantages if the lightsources are small [SPNP96]. The problem of point lightsources can be solved by a modified version of the "**first-shot**" that shoots the power of the point lightsources onto other surfaces, then removes them from the scene [CMS98].

This is a little bit more complicated for ray-bundle tracing which handles also non-diffuse reflection. Since the surfaces can also be non-diffuse, the incoming radiance received by the patches from each point lightsource should be stored (this requires l additional variables per patch, where l is the number of point lightsources). The secondary, non-diffuse emission to a direction is computed from these irradiances.

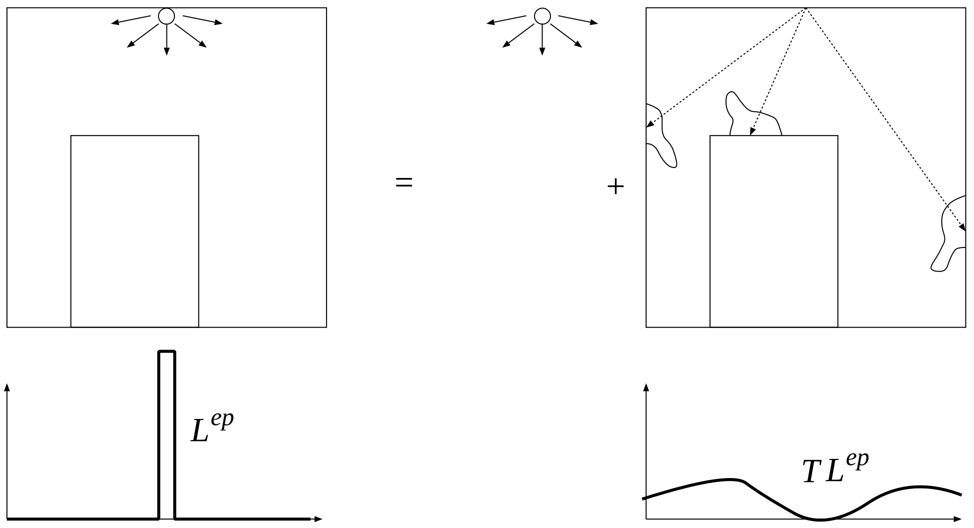


Figure 8.22: First shot technique

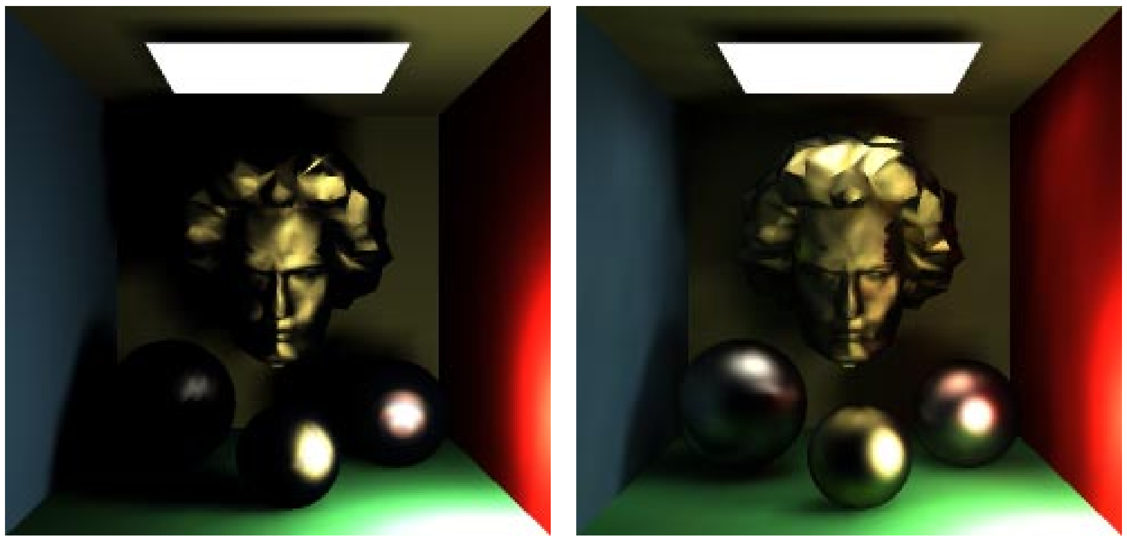


Figure 8.23: A scene after the "first-shot"(left) and after 500 global ray-bundle walks (right)

Chapter 9

Iteration solution of the global illumination problem

Iteration techniques realize that the solution of the rendering equation is the fixed point of the following iteration scheme

$$L_m = L^e + \mathcal{T} L_{m-1}, \tag{9.1}$$

and obtain the pixel colors as a limiting value:

$$\mathcal{M}L = \lim_{n\to\infty} \mathcal{M}L_m. \tag{9.2}$$

Since iteration uses the complete radiance function, its temporary versions should be stored. To do this with only finite data, finite-element techniques can be used. Recall that these finite-element techniques approximate the unknown function in the following finite function series form:

$$L(\vec{x},\omega) \approx L^{(n)}(\vec{x},\omega) = \sum_{j=1}^{n} \mathbf{L}_j \cdot b_j(\vec{x},\omega) = \mathbf{b}^T(\vec{x},\omega) \cdot \mathbf{L} \tag{9.3}$$

where $b_j(\vec{x},\omega)$ is a system of predefined basis functions, and $\mathbf{L}_j$ factors are unknown coefficients.

Substituting this approximation into the iteration of the rendering equation we obtain:

$$\sum_{j=1}^{n} \mathbf{L}_j(m) \cdot b_j(\vec{x},\omega) \approx \sum_{j=1}^{n} \mathbf{L}^e_j \cdot b_j(\vec{x},\omega) + \mathcal{T} \sum_{j=1}^{n} \mathbf{L}_j(m-1) \cdot b_j(\vec{x},\omega) = \sum_{j=1}^{n} \mathbf{L}^e_j \cdot b_j(\vec{x},\omega) + \sum_{j=1}^{n} \mathbf{L}_j(m-1) \cdot \mathcal{T} b_j(\vec{x},\omega). \tag{9.4}$$

Note that equality cannot be guaranteed, since even if $\sum_{j=1}^{n} \mathbf{L}_j \cdot b_j(\vec{x},\omega)$ is in the subspace defined by the basis functions, the integral operator $\mathcal{T}$ may result in a function that is out of this space. Instead, equality is required in an appropriate subspace defined by **adjoint basis** functions $\tilde{b}_1(\vec{x}), \tilde{b}_2(\vec{x}), \ldots \tilde{b}_n(\vec{x})$ (chapter 5).

Having multiplied the iteration formula by each adjoint basis functions $\tilde{b}_i$, we obtain the following iteration solution of a system of linear equations:

$$\mathbf{L}_i(m) = \mathbf{L}^e_i + \sum_{j=1}^{n} \langle \mathcal{T} b_j, \tilde{b}_i \rangle \cdot \mathbf{L}_j(m-1). \tag{9.5}$$

This system of linear equations can also be presented in a vector form:

$$\mathbf{L}(m) = \mathbf{L}^e + \mathbf{R} \cdot \mathbf{L}(m-1), \qquad \mathbf{R}_{ij} = \langle \mathcal{T} b_j, \tilde{b}_i \rangle. \tag{9.6}$$

The complexity of the iteration solution depends on the number of operations needed for a single iteration step and the number of iterations providing convergence. A single step of the iteration requires the multiplication of an n dimensional vector and an $n \times n$ dimensional matrix (where n is the number of basis functions), which requires $\mathcal{O}(n^2)$ operations.

Concerning the number of necessary steps, we concluded in section 4.3.1 that the speed of convergence is at least geometric by a factor $\lambda = ||\mathbf{R}||$. Thus the number of steps to decrease the error below a predefined limit depends only on λ and is independent of n. To explain this, let us examine the diffuse radiosity case, when piece-wise constant basis functions are used, and thus matrix $\mathbf{R}$ contains the patch-to-patch form factors weighted by the albedos of patches. The infinite norm of $\mathbf{R}$ is close to being independent of the number of surface elements, since as the number of surface elements increases, the value of form factors decreases, sustaining a constant sum of rows. This constant sum represents that portion of the energy radiated by surface i, which is gathered by other surfaces, multiplied by the albedo of surface i. Consequently, the number of necessary iterations is independent of the number of surface elements, making the iteration solution an $\mathcal{O}(n^2)$ process.

More sophisticated techniques, such as the **Gauss-Seidel iteration**, **overrelaxation**, **Southwell-iteration** can either speed-up the process or reduce the storage requirements, but they cannot change the quadratic complexity [SKM95].

9.1 Why should we use Monte-Carlo iteration methods?

Deterministic iteration has two critical problems. On the one hand, since the domain of the radiance function is 4 dimensional and has usually high variation, an accurate finite-element approximation often requires very many basis functions, which, in turn, need a lot of storage space. On the other hand, when finite element techniques are applied, the transport operator is only approximated, which introduces some non-negligible error in each step. As concluded in section 4.3.1, if the contraction ratio of the operator is λ, then the total accumulated error will be approximately $1/(1-\lambda)$ times the error of a single step [SKFNC97]. For highly reflective scenes, the iteration is slow and the result is inaccurate if the approximation of the operator is not very precise. Very accurate approximations of the transport operator, however, require a lot of computation time and storage space.

Both problems can be successfully attacked by randomizing the iteration, which is called the **stochastic iteration**. The basic idea of stochastic iteration is that instead of approximating operator $\mathcal{T}$ in a deterministic way, a much simpler random operator is used during the iteration, which "behaves" as the real operator just in the "average" case.

When stochastic iteration is applied, the transport operator should be like the real operator just in the average case, the method will still converge to the real solution. Thus the error accumulation problem can also be avoided. This also allows us to use significantly simpler realizations. For example, the integral part of the operator can also be interpreted as an expected value, thus in a single transfer usually no explicit integral is computed. As we shall see, it is relatively easy to find random operators whose expected case behavior matches exactly to that of the real operator.

On the other hand, if the operator is carefully randomized, it does not require the integrand everywhere in the domain, which allows us not to store the complete radiance function, thus a lot of storage space can be saved. Compared to the astronomical storage requirements of non-diffuse radiosity methods, for example, with stochastic iteration we can achieve the same goal with one variable per patch [SKP98]. This argument loses some of its importance when view-independent solution is also required, since the final solution should be stored anyway. This is not a problem if only the diffuse case is considered, since using a single radiosity value per patch the image can be generated from any viewpoint. For the non-diffuse case, the reduced storage gets particularly useful when the image is to be calculated in only a single, or in a few eye positions.

Summarizing, the advantages of stochastic iteration are the simplicity, speed, affordable storage requirements and numerical stability even for very large systems containing highly reflective materials.

9.2 Formal definition of stochastic iteration

The concept of stochastic iteration has been proposed and applied for the diffuse radiosity problem in [Neu95, NFKP94, NPT$^+$95, SKFNC97], that is for the solution of finite-dimensional linear equations.

In this section we generalize the fundamental concepts to solve integral equations [SK98b, SK99b], then the generalized method will be used for attacking non-diffuse global illumination problems.

Suppose that we have a random linear operator $\mathcal{T}^*$ so that

$$E[\mathcal{T}^* L] = \mathcal{T} L \tag{9.7}$$

for any Riemann-integrable function L.

During stochastic iteration a random sequence of operators $\mathcal{T}_1^*, \mathcal{T}_2^*, \ldots, \mathcal{T}_i^*, \ldots$ is generated, which are instantiations of $\mathcal{T}^*$, and this sequence is used in the iteration:

$$L_m = L^e + \mathcal{T}_m^* L_{m-1}. \tag{9.8}$$

Since in computer implementations the calculation of a random operator may invoke finite number of random number generator calls, we are particularly interested in those random operators which have the following construction scheme:

1. Random "point" p_i is found from a finite dimensional set Π using probability density $\mathrm{prob}(p)$. This probability density may or may not depend on function L.

2. Using p_i a "deterministic" operator $\mathcal{T}^*(p_i)$ is applied to radiance L.

Point p_i is called the **randomization point** since it is responsible for the random nature of operator $\mathcal{T}^*$.

Using a sequence of random transport operators, the measured value

$$P_m = \mathcal{M}L_m \tag{9.9}$$

will also be a random variable which does not converge but fluctuates around the real solution. However, as it will be shown, the solution can be found by averaging the estimates of the subsequent iteration steps.

Formally the sequence of the measured values during the iteration is the following:

$$\begin{aligned}
P_1 &= \mathcal{M}L_1 = \mathcal{M}(L^e + \mathcal{T}_1^* L^e), \\
P_2 &= \mathcal{M}L_2 = \mathcal{M}(L^e + \mathcal{T}_2^* L^e + \mathcal{T}_2^* \mathcal{T}_1^* L^e), \\
&\vdots \\
P_M &= \mathcal{M}L_M = \mathcal{M}(L^e + \mathcal{T}_M^* L^e + \mathcal{T}_M^* \mathcal{T}_{M-1}^* L^e + \mathcal{T}_M^* \mathcal{T}_{M-1}^* \mathcal{T}_{M-2}^* L^e + \ldots).
\end{aligned}$$

Averaging the first M steps, we obtain:

$$\tilde{P}_M = \frac{1}{M}\sum_{i=1}^{M} \mathcal{M}L_i = \mathcal{M}(L^e + \frac{1}{M}\sum_{i=1}^{M} \mathcal{T}_i^* L^e + \frac{1}{M}\sum_{i=1}^{M-1} \mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e + \frac{1}{M}\sum_{i=1}^{M-2} \mathcal{T}_{i+2}^* \mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e + \ldots) =$$

$$\mathcal{M}(L^e + \frac{1}{M}\sum_{i=1}^{M} \mathcal{T}_i^* L^e + \frac{M-1}{M} \cdot \frac{1}{M-1}\sum_{i=1}^{M-1} \mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e + \frac{M-2}{M} \cdot \frac{1}{M-2}\sum_{i=1}^{M-2} \mathcal{T}_{i+2}^* \mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e + \ldots). \tag{9.10}$$

In order to prove that $\tilde{P}_M$ really converges to the solution of the integral equation, first it is shown that the expected value of $\mathcal{T}_{i+k}^* \mathcal{T}_{i+k-1}^* \ldots \mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e$ is $\mathcal{T}^{k+1} L^e$. For $k = 0$, it comes directly from the requirement of equation (9.7). For $k = 1$, the **total expected value theorem** [Rén62] can be applied:

$$E[\mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e] = \int_{\Pi} E[\mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e | p_{i+1} = p] \cdot \mathrm{prob}(p)\, dp. \tag{9.11}$$

Since for a fixed $p_{i+1} = p$, operator $\mathcal{T}_{i+1}^*$ becomes a deterministic linear operator, its order can be exchanged with that of the expected value operator:

$$E[\mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e | p_{i+1} = p] = \mathcal{T}_{i+1}^*(p) \left(E[\mathcal{T}_i^* L^e]\right). \tag{9.12}$$

Using requirement (9.7), the expected value $E[\mathcal{T}_i^* L^e]$ is $\mathcal{T}L^e$, thus we further obtain

$$E[\mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e | p_{i+1} = p] = \mathcal{T}_{i+1}^*(p)(\mathcal{T}L^e). \tag{9.13}$$

Substituting this to equation (9.11), we get

$$E[\mathcal{T}_{i+1}^* \mathcal{T}_i^* L^e] = \int_{\Pi} \mathcal{T}_{i+1}^*(p)(\mathcal{T}L^e) \cdot \mathrm{prob}(p)\, dp = E[\mathcal{T}_{i+1}^*(\mathcal{T}L^e)] = \mathcal{T}(\mathcal{T}L^e) = \mathcal{T}^2 L^e, \tag{9.14}$$

which concludes our proof for the $k = 1$ case. The very same idea can be used recursively for more than two terms.

Returning to the averaged solution $\tilde{P}_M$, its expected value is then

$$E[\tilde{P}_M] = \mathcal{M}(L^e + \mathcal{T}L^e + \frac{M-1}{M}\mathcal{T}^2 L^e + \frac{M-2}{M}\mathcal{T}^3 L^e + \ldots + \frac{1}{M}\mathcal{T}^M L^e). \tag{9.15}$$

Note also that there is some power "defect" ΔP between this expected value and the real solution

$$\mathcal{M}(L^e + \mathcal{T}L^e + \mathcal{T}^2 L^e + \mathcal{T}^3 L^e + \ldots)$$

because of the missing higher order terms for finite M values. Denoting the contraction ratio of the integral operator $\mathcal{T}$ by λ, and assuming that the measuring device is calibrated to show value c for unit homogeneous radiance, this defect can be estimated as follows:

$$|\Delta P| = \left| \mathcal{M}\left(\frac{1}{M}\mathcal{T}^2 L^e + \frac{2}{M}\mathcal{T}^3 L^e + \ldots \frac{M-1}{M}\mathcal{T}^M L^e + \mathcal{T}^{M+1} L^e + \mathcal{T}^{M+2} L^e + \ldots \right) \right| \le$$

$$\frac{c\lambda^2}{M} \cdot ||L^e|| \cdot (1 + 2\lambda + 3\lambda^2 + \ldots + (M-1)\lambda^{M-2} + M\lambda^{M-1} + M\lambda^M + \ldots) =$$

$$\frac{c\lambda^2}{M} \cdot ||L^e|| \cdot \left[\frac{d}{d\lambda} \left(\sum_{i=1}^{M-1} \lambda^i \right) + M \cdot \frac{\lambda^{M-1}}{1-\lambda} \right] \leq \frac{c}{M} \cdot \frac{\lambda^2}{(1-\lambda)^2} \cdot ||L^e||. \tag{9.16}$$

This defect converges to zero if the operator is a contraction and M goes to infinity, thus we have

$$\lim_{M\to\infty} E[\tilde{P}_M] = \mathcal{M}(L^e + \mathcal{T}L^e + \mathcal{T}^2 L^e + \mathcal{T}^3 L^e + \ldots). \tag{9.17}$$

We mention that the defect can further be reduced, thus the convergence can be improved, by ignoring the first few iterations in the averaged result [Neu95, Sbe96].

Finally, it must be explained why and when random variable $\tilde{P}_M$ converges to its expected value. Looking at formula (9.10) we can realize that it consists of sums of the following form:

$$\frac{1}{M-k} \cdot \sum_{i=1}^{M-k} \mathcal{T}^*_{i+k} \mathcal{T}^*_{i+k-1} \ldots \mathcal{T}^*_{i+1} \mathcal{T}^*_i L^e.$$

According to the theorems of large numbers, and particularly to the Bernstein [Rén62] theorem, these averages really converge to the expected value if the terms in the average are not highly correlated (note that here the terms are not statistically independent as assumed by most of the laws of large numbers). It means that random variables $\mathcal{T}^*_{i+k} \mathcal{T}^*_{i+k-1} \ldots \mathcal{T}^*_i L^e$ and $\mathcal{T}^*_{j+k} \mathcal{T}^*_{j+k-1} \ldots \mathcal{T}^*_j L^e$ should not have strong correlation if $i \neq j$. This is always true if the operators are generated from independent random variables.

To show a negative example, let us assume that there is a very strong correlation between the random operators, namely that different $\mathcal{T}^*_i$ random operators use exactly the same randomization point. When this randomization point is known, the iteration is fully deterministic using operator $\mathcal{T}^*_i = \mathcal{T}^*$. The limiting value of this iteration will be a random variable

$$\mathcal{M}(L^e + (\mathcal{T}^*)L^e + (\mathcal{T}^*)^2 L^e + (\mathcal{T}^*)^3 L^e + \ldots),$$

which usually differ from the expected solution.

There is another danger that should be considered. Random variable $\tilde{P}_M$ is a sum of random variables whose number goes to infinity. Even if the variances of all single terms in this sum converge to zero, the variance of the sum of these terms might still converge to a non-zero value. In the context of random walks, this phenomenon is called "non-existing variance" [Sbe99, Erm75]. To avoid this case, random operators should not be "*over-randomized*", thus its variance must not exceed a certain limit.

9.2.1 Other averaging techniques

In the previous section we solved the problem that stochastic iteration is not convergent by simply averaging the values generated during iteration. There are other averaging schemes, on the other hand, that use even more combinations of the preceding random operators.

Progressive Monte-Carlo

Progressive Monte-Carlo uses the following formulae to derive a new value from the previous one:

$$\begin{aligned} L'_m &= L^e + \mathcal{T}^*_m L_{m-1}, \\ L_m &= \tau_m \cdot L'_m + (1-\tau_m) \cdot L_{m-1}, \\ \tilde{P}_m &= \mathcal{M} L_n, \end{aligned} \tag{9.18}$$

where τ_m is an appropriate sequence that converges to 0, as for example, $\tau_m = 1/m$.

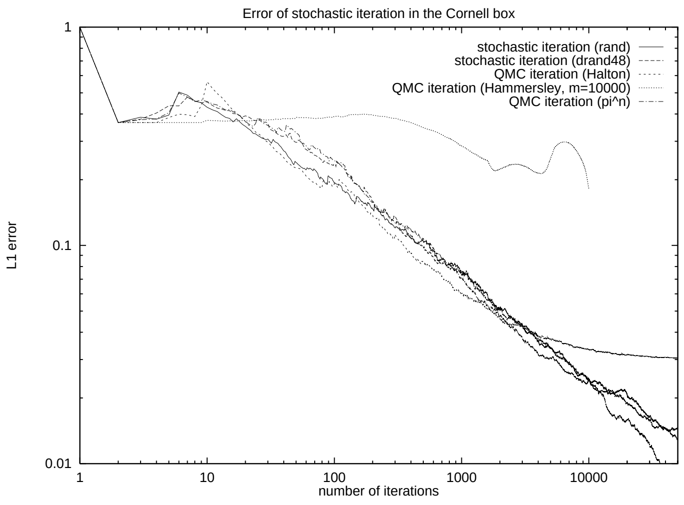
To allow comparison, the corresponding formulae of the normal iteration are also presented here:

$$\begin{aligned} L_m &= L^e + \mathcal{T}^*_m L_{m-1}, \\ \tilde{P}_m &= \tau_m \cdot \mathcal{M} L_m + (1-\tau_m) \cdot P_{m-1}. \end{aligned} \tag{9.19}$$

Note that the fundamental difference is that progressive Monte-Carlo uses the average of the previous samples not only in the final estimate but also to continue iteration. Progressive Monte-Carlo thus can use all combinations of the preceding random operators to compute the actual result. However, it also has energy defect.

9.2.2 Can we use quasi-Monte Carlo techniques in iteration?

Stochastic iteration can also be viewed as a single walk which uses a single sequence of usually 4-dimensional randomization points (for ray-bundle tracing 2-dimensional randomization points), and the $\mathcal{T}^*_{i+k} \mathcal{T}^*_{i+k-1} \ldots \mathcal{T}^*_i L^e$ terms are included in integral quadratures simultaneously for all k.



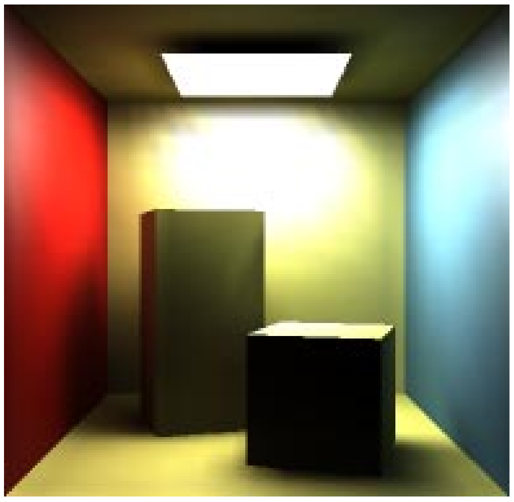


Figure 9.1: Ray-bundle based stochastic iteration with random and quasi-random numbers and the used test scene

It means that the randomization points should support not only 4-dimensional integration, but using subsequent pairs also 8-dimensional integration, using the subsequent triplets 12-dimensional integration, etc. Sequences that support k-dimensional integrals when subsequent k-tuples are selected are called **k-uniform sequences** [Knu81]. The widely used Halton or Hammersley sequences are only 1-uniform, thus theoretically they should provide false results.

This is obvious for the Hammersley sequence, in which the first coordinate is increasing. Such a sequence would search for only those multiple reflections where the angle corresponding to the first coordinate always increases in subsequent reflections.

It is less obvious, but is also true for the Halton sequence. Due to its construction using radical inversion, the subsequent points in the sequence are rather far, thus only those reflections are considered, where the respective angle changes drastically.

In order to avoid this problem without getting rid of the quasi-Monte Carlo sequences, [SKFNC97] proposed the random scrambling of the sample points. The same problem arises, for example, when generating uniform distributions on a sphere, for which [CMS98] proposed to increase the dimension of the low-discrepancy sequence.

Note that this problem is specific to quasi-Monte Carlo integration and does not occur when classical Monte-Carlo method is used to select the sample points (a random sequence is ∞-uniform [Knu81]).

In order to demonstrate these problems, we tested the ray-bundle based iteration for different random (i.e. pseudo-random) and low-discrepancy sequences. The test scene was the Cornell box. In figure 9.1 we can see that the Hammersley sequence gives completely wrong result and the Halton sequence also deteriorates from the real solution. The two random generators (rand and drand48), however, performed well.

The figure also included a modification of the $q_m = \{\pi^m\}$ quasi-Monte Carlo sequence (operator $\{\}$ selects the fractional part of a number). This is believed to be (but has not been proven to be) ∞-uniform [Deá89]. However, this sequence is very unstable numerically, therefore we used the following scheme started at $\kappa_0 = 1$:

```
$\kappa_m = \kappa_{m-1} \cdot (\pi - 2)$
**if** $\kappa_m > 100000$ **then** $\kappa_m = \kappa_m - 100000$
$q_m = \{\kappa_m\}$
```

Chapter 10

Review of stochastic iteration algorithms

In order to use stochastic iteration in practice, the key problem is the definition of the random transport operator. This operator should meet the requirement of equation (9.7) and should be easy to compute.

For the continuous case, a single application of the transport operator contains a directional integral. For the finite element case, the transport operator also includes the projection to the adjoint base which requires additional integration in the domain of basis functions.

Following the general concepts of Monte-Carlo methods, we usually do not intend to compute the integrals **explicitly**, but want to get them as an expected value. Thus the different random transport operators can be classified according to which integrals are evaluated explicitly using some deterministic quadrature and which integrals are computed **implicitly** as an expected value.

First diffuse radiosity algorithm are presented. The diffuse radiosity method is a finite-element approach which takes advantage of the fact that a patch can be represented by a single radiance (or radiosity) value. Thus the rendering equation is converted to a system of linear equations, where the number of unknowns is equal to the number of patches. When the non-diffuse case is considered, then the storage complexity becomes a critical issue. Therefore, in the second part of this chapter, we discuss how the required storage can be reduced in a flexible way by stochastic iteration.

10.1 Stochastic iteration for the diffuse radiosity

In the gathering type radiosity algorithms the projected rendering equation (formula (5.5)) has the following form

$$\mathbf{L} = \mathbf{L}^e + \mathbf{R} \cdot \mathbf{L}.$$

Alternatively, shooting radiosity algorithms are based on the projected potential equation (formula (5.8)):

$$\mathbf{P} = \mathbf{P}^e + \mathbf{H} \cdot \mathbf{P}.$$

According to the basic requirement of stochastic iteration we need to find random operators $\mathcal{T}_F^*$ or $\mathcal{T}_F^{'*}$ that behave as the real operator in average, that is

$$E[\mathcal{T}_F^* \mathbf{L}] = \mathbf{R} \cdot \mathbf{L},$$

$$E[\mathcal{T}_F^{'*} \mathbf{P}] = \mathbf{H} \cdot \mathbf{P}. \tag{10.1}$$

The evaluation of $(\mathbf{R} \cdot \mathbf{L})_i$ or alternatively of $(\mathbf{H} \cdot \mathbf{P})|_i$ requires a surface and a directional integration (or in other formulations two surface integrations).

The possible alternatives for a random transport operator are as follows:

1. Both integrals are explicitly computed but only for a randomly selected subset of the patches.

2. The surface integral explicitly computed but the directional integral implicitly.

3. Compute the surface integral implicitly but the directional integral explicitly. This method can, for example, use hemicubes for the directional integration but selects the center of the hemicube randomly on the patch.

4. Both integrals are computed implicitly.

10.1.1 Stochastic radiosity

In **stochastic radiosity** [NFKP94], the randomized operator is simplified in a sense that it first selects a single (or a few) patches with probability proportional to their power and then calculates the transfer only from this important patch as if it had all the power $\Phi = \sum_{k=1}^{n} \mathbf{P}_k$. Thus here both integrals are explicitly computed but only for a subset of patches.

To prove that it meets the requirement stated by equation (10.1), let us suppose that patch j has been selected and let us examine the new power of patch i:

$$(\mathcal{T}_F^{\prime *}\mathbf{P})|_i = \mathbf{H}_{ij} \cdot \Phi. \tag{10.2}$$

Since the probability of selecting patch j is $\mathbf{P}_j/\Phi$, the expectation of the new power is

$$E[(\mathcal{T}_F^{\prime *}\mathbf{P})|_i] = \sum_{j=1}^{n} \mathbf{H}_{ij} \cdot \Phi \cdot \frac{\mathbf{P}_j}{\Phi} = \sum_{j=1}^{n} \mathbf{H}_{ij} \cdot \mathbf{P}_j \tag{10.3}$$

which we wanted to prove.

10.1.2 Transillumination radiosity

The **transillumination radiosity method** [Neu95, SKFNC97] has also a stochastic iteration version. It defines the random transport operator by uniformly selecting D **transillumination directions** $\omega_1', \dots \omega_D'$ and allowing patches to interact only in these transillumination directions. In order to calculate these interactions, a large discretized window is placed perpendicularly to each transillumination direction and the radiance transfer to a patch is approximated by elementary transfers going through the pixels covering the projection of the patch.

Let us consider a single transillumination direction. Projecting patch A_i onto a plane that is perpendicular to the transillumination direction and then approximating the integral of the incoming radiance here by a discrete sum, we get

$$\int_{A_i} L(h(\vec{x}, -\omega_d')) \cdot \cos\theta_d' \, d\vec{x} = \int_{A_i^p} L(h(\vec{x}', -\omega_d')) \cdot \, d\vec{x}' \approx \sum_{P \in A_i^p} \mathbf{L}_{\mathrm{buffer_d}[P]} \cdot \delta A. \tag{10.4}$$

where $\mathrm{buffer_d}[P]$ stores the index of that patch which is visible in pixel P in the transillumination direction ω_d' from patch i, and δA is the size of a pixel of the buffer (figure 10.1).

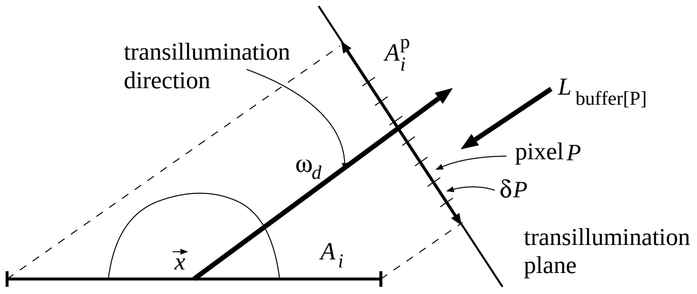


Figure 10.1: Integration on the transillumination plane

Thus the random transfer operator is

$$(\mathcal{T}_F^{*}\mathbf{L})|_i = \frac{4\pi \cdot f_i \cdot \delta A}{D} \sum_{d=1}^{D} \sum_{P \in A_i^p} \mathbf{L}_{\mathrm{buffer_d}[P]}. \tag{10.5}$$

If the transillumination directions are uniformly distributed and the buffer is uniformly jittered, then the expected value of this operator is

$$E[(\mathcal{T}_F^{*}\mathbf{L})|_i] = \int_{\Omega}\int_{P} \frac{4\pi \cdot f_i \cdot \delta A}{D} \sum_{d=1}^{D} \sum_{P \in A_i^p} \mathbf{L}_{\mathrm{buffer_d}[P]} \, \frac{dp}{\delta A} \, \frac{d\omega_d'}{4\pi} = \frac{1}{D}\sum_{d=1}^{D} \int_{\Omega}\int_{P} \sum_{P \in A_i^p} \mathbf{L}_{\mathrm{buffer_d}[P]} \cdot f_i \, dp \, d\omega_d'.$$

If uniform jittering is applied, then we can usually assume that the discrete approximation of the positional radiance distribution gives back the real distribution in the average case, that is

$$\int_{P} \sum_{P \in A_i^p} \mathbf{L}_{\mathrm{buffer_d}[P]} \, dp = \int_{A_i^p} L(h(\vec{x}', -\omega_d')) \, d\vec{x}'. \tag{10.6}$$

However, this statement is not always true if incremental polygon filling algorithms are applied [SK98a, SK98b]. Note, for example, that an incremental polygon filling algorithm always generates an approximation whose width and height are at least 1. Thus this assumption is correct if the resolution of the discrete buffer is high enough to project each polygon onto at least a few pixels. Substituting this to the expected value integral, we get

$$E[(\mathcal{T}_F^*\mathbf{L})|_i] = \frac{1}{D}\sum_{d=1}^{D}\int_{\Omega}\int_{A_i^p} L(h(\vec{x}', -\omega_d'))\, d\vec{x}' \cdot f_i\, d\omega_d' = \int_{\Omega}\int_{A_i} L(h(\vec{x}', -\omega')) \cdot f_i \cdot \cos\theta'\, d\vec{x} d\omega'. \tag{10.7}$$

Using $L(h(\vec{x}', -\omega')) = \sum_{j=1}^{n} b_j(h(\vec{x}', -\omega')) \cdot \mathbf{L}_j$ and equation (5.21), we can prove that the expectation really gives back the real projected transport operator:

$$E[(\mathcal{T}_F^*\mathbf{L})|_i] = \sum_{j=1}^{n}\int_{\Omega}\int_{A_i} b_j(h(\vec{x}', -\omega')) \cdot f_i \cdot \cos\theta_d'\, d\vec{x} d\omega' \cdot \mathbf{L}_j = \sum_{j=1}^{n}\mathbf{R}_{ij} \cdot \mathbf{L}_j. \tag{10.8}$$

10.1.3 Stochastic ray-radiosity

Stochastic ray-radiosity [NPT$^+$95] approximates the transport operator by M random rays that are sampled proportionally to the power of the patches. On a patch the starting point of the ray is sampled using a uniform distribution, while the direction follows a cosine distribution. A single ray carries Φ/M power. Thus this method approximates both integrals implicitly.

Let us examine the case when a single ray is selected (since different rays are sampled from the same distribution, the effect of M rays will be M times the effect of a single ray in the expected value). Suppose that patch j is selected as a shooting patch. The probability of the selection event is $\mathbf{P}_j/\Phi$. Thus the probability density of selecting a point $\vec{x}$ of a patch and a direction ω is

$$\frac{\mathbf{P}_j}{\Phi} \cdot \frac{1}{A_j} \cdot \cos\theta.$$

This transfers Φ/M power to the patch that is hit by the ray where the reflected power is computed. Thus the random transport operator for a single ray is

$$E[(\mathcal{T}_F^{'*}\mathbf{P})|_i] = M \cdot f_i \cdot \sum_{j=1}^{n}\int_{A_j}\int_{\Omega} b_i(h(\vec{y}, \omega)) \cdot \frac{\Phi}{M} \cdot \frac{1}{A_j} \cdot \cos\theta\, d\vec{y} d\omega \cdot \frac{\mathbf{P}_j}{\Phi} =$$

$$\sum_{j=1}^{n}\frac{f_i}{A_j} \cdot \int_{A_j}\int_{\Omega} b_i(h(\vec{y}, \omega)) \cdot \cos\theta\, d\vec{y} d\omega \cdot \mathbf{P}_j = \sum_{j=1}^{n}\mathbf{H}_{ij} \cdot \mathbf{P}_j. \tag{10.9}$$

10.2 Definition of the random transport operator for the non-diffuse finite-element case

When moving towards the non-diffuse case, another requirement must be imposed upon the random transport operator. It must not only meet the requirement of equation (9.7) and be easy to compute, but it must also allow the compact representation of the $\mathcal{T}_i^* L$ functions. This extra requirement is evident if we take into account that unlike in the diffuse case, the domain of L is a 4-dimensional continuous space, so is the domain of $\mathcal{T}_i^* L$. From the point of view of compactness, what we have to avoid is the representation of these functions over the complete domain.

Thus those transport operators are preferred, which require the value of L just in a few "domain points" (e.g. in a single "domain point"). Note that the evaluation of $\mathcal{T}_i^* L$ now consists of the following steps: first a randomization point p_i is found to define random operator $\mathcal{T}_i^*$, which in turn determines at which domain point the value of L is required. Up to now, we have had complete freedom to define the set of randomization points. One straightforward way is defining this set to be the same as (or a superset of) the domain of the radiance function and using random transport operators that require the value of the radiance function at their randomization points. Although this equivalence is not obligatory, it can significantly simplify the computations, since when the randomization point is generated, the required domain point is also known.

Using random operators that evaluate the radiance in a single point is not enough in itself, since even a single "point" can result in a continuous $\mathcal{T}_i^* L$ function, which must be stored and re-sampled in the subsequent iteration step and also by the measurement. The solution is the postponing of the complete calculation of $\mathcal{T}_i^* L$ until it is known where its value is needed in the next iteration step and by the measuring device. In this way, the random operator should be evaluated twice but just for two points. Once for the actual and the previous "points" resulting in

$[\mathcal{T}^*(p_i)L(p_i)](p_{i+1})$, and once for $p_{\rm eye}$ which is needed by the measuring device and for previous point providing $[\mathcal{T}^*(p_i)L(p_i)](p_{\rm eye})$.

The complete iteration goes as follows:

```
P = 0                                                        // initialize the measured value to zero
Find $p_1$ randomly                                          // select the randomization point of the first iteration
$L(p_1) = L^e(p_1)$
for $i = 1$ to $M$ do
    $P^{\rm new} = L^e(p_{\rm eye}) + [\mathcal{T}^*(p_i)L(p_i)](p_{\rm eye})$        // measure the radiance
    $P = \mathcal{M}P^{\rm new} \cdot 1/i + (1 - 1/i) \cdot P$                          // average the measured value
    Find $p_{i+1}$ randomly                                  // select the randomization point of the next iteration
    $L(p_{i+1}) = L^e(p_{i+1}) + [\mathcal{T}^*(p_i)L(p_i)](p_{i+1})$  // a single step of stochastic iteration
endfor
Display final image
```

10.2.1 Single ray based transport operator

The continuous formulation has just a single directional integral, thus a random transport operator can evaluate this single integral implicitly. This results in a method that uses a “single” random walk to obtain the solution.

More specifically, let the random transport operator use a single ray having random origin $\vec{y}_i$ and direction ω_i generated with a probability that is proportional to the cosine weighted radiance of this point at the given direction.

This ray transports the whole power

$$\Phi = \int_S \int_\Omega L(\vec{y}, \omega') \cos\theta_{\vec{y}} \, d\omega' \, d\vec{y}$$

to that point $\vec{x}$ which is hit by the ray. Formally, the random transport operator is

$$(\mathcal{T}^* L)(\vec{x}, \omega) = \Phi \cdot \delta(\vec{x} - h(\vec{y}, \omega_i)) \cdot f_r(\omega_i, \vec{x}, \omega). \tag{10.10}$$

Let us prove that this random operator meets the requirement of equation (9.7). The probability density of selecting surface point $\vec{y}$ and direction ω' is

$$\frac{d\,\Pr\{\vec{y}, \omega'\}}{d\vec{y}\, d\omega_{\vec{y}}} = \frac{L(\vec{y}, \omega') \cdot \cos\theta_{\vec{y}}}{\Phi} \tag{10.11}$$

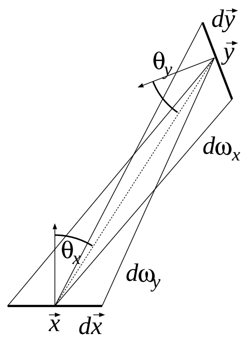


Figure 10.2: Symmetry of solid angles of shooting and gathering

Using the definition of the solid angle

$$d\omega_{\vec{y}} = \frac{d\vec{x} \cdot \cos\theta'_{\vec{x}}}{|\vec{y} - \vec{x}|^2}$$

we can obtain a symmetry relation (figure 10.2) for the shooting and gathering solid angles:

$$d\vec{y} \cdot d\omega_{\vec{y}} \cdot \cos\theta_{\vec{y}} = d\vec{y} \cdot \frac{d\vec{x} \cdot \cos\theta'_{\vec{x}}}{|\vec{y} - \vec{x}|^2} \cdot \cos\theta_{\vec{y}} = d\vec{x} \cdot \frac{d\vec{y} \cdot \cos\theta_{\vec{y}}}{|\vec{y} - \vec{x}|^2} \cdot \cos\theta'_{\vec{x}} = d\vec{x} \cdot d\omega'_{\vec{x}} \cdot \cos\theta'_{\vec{x}}. \tag{10.12}$$

Thus the probability of selecting $\vec{y}, \omega'$ can also be expressed in the following way:

$$d\,\Pr\{\vec{y}, \omega'\} = \frac{L(\vec{y}, \omega') \cdot \cos\theta_{\vec{y}}}{\Phi} \cdot d\vec{y}\, d\omega_{\vec{y}} = \frac{L(h(\vec{x}, -\omega'), \omega') \cdot \cos\theta_{\vec{x}}}{\Phi} \cdot d\vec{x}\, d\omega'_{\vec{x}}. \tag{10.13}$$

Now we can easily prove that the random transport operator meets requirement (9.7) since

$$E[(\mathcal{T}^* L)(\vec{x},\omega)] = \int_S \int_\Omega \Phi \cdot \delta(\vec{x} - h(\vec{y},\omega')) \cdot f_r(\omega',\vec{x},\omega)\, d\Pr\{\vec{y},\omega'\} =$$

$$\int_\Omega L(h(\vec{x},-\omega'),\omega') \cdot \cos\theta'_{\vec{x}} \cdot f_r(\omega',\vec{x},\omega)\, d\omega'_{\vec{x}} = (\mathcal{T}L)(\vec{x},\omega). \tag{10.14}$$

Note that the result of the application of the random operator can be a single point that receives all the power and reflects some part of it or the result can be no point at all if the ray leaves the scene.

Suppose that the first random operator $\mathcal{T}_1^*$ is applied to L^e which may transfer all power

$$\Phi_1 = \int_S \int_\Omega L^e(\vec{y}_1,\omega_1) \cos\theta_{\vec{y}_1}\; d\omega_1\; d\vec{y}_1$$

to a single point $\vec{x}_1 = h(\vec{y}_1,\omega_1)$ using probability density

$$\frac{d\Pr_1\{\vec{y}_1,\omega_1\}}{d\vec{y}_1 d\omega_1} = \frac{L^e(\vec{y}_1,\omega_1)\cdot\cos\theta_{\vec{y}_1}}{\Phi}.$$

Before continuing with the second step of the iteration, the radiance should be measured, that is, an image estimate should be computed from $L^e + \mathcal{T}_1^* L^e$. We can separately calculate the effect of the lightsources on the image and then add the effect of $\mathcal{T}_1^* L^e$. Note that $\mathcal{T}_1^* L^e$ is concentrated in a single point, thus its contribution can be computed by tracing a ray from the eye to this point, and if this point is not occluded, then adding $f_r(\omega_1,\vec{x},\omega_{\rm eye})\cdot\Phi$ to that pixel in which $\vec{x}$ is visible.

The second operator $\mathcal{T}_2^*$ should be applied to

$$L_1 = L^e + \mathcal{T}_1^* L^e,$$

thus both the total power Φ and the probability density have been modified:

$$\Phi_2 = \int_S \int_\Omega L_1(\vec{y}_2,\omega_2) \cos\theta_{\vec{y}_2}\; d\omega_2\; d\vec{y}_2 = \Phi_1 \cdot (1 + a_{\vec{x}_1}(\omega_1))$$

where $a_{\vec{x}_1}$ is the **albedo** at point $\vec{x}_1$ defined by

$$a_{\vec{x}}(\omega) = \int_\Omega f_r(\omega,\vec{x},\omega') \cos\theta'_{\vec{x}}\; d\omega',$$

and the new probability density is

$$\frac{d\Pr_2\{\vec{y}_2,\omega_2\}}{d\vec{y}_2 d\omega_2} = \frac{L_1(\vec{y}_2,\omega_2)\cdot\cos\theta_{\vec{y}_2}}{\Phi} = \frac{L^e(\vec{y}_2,\omega_2)\cdot\cos\theta_{\vec{y}_2} + f_r(\omega_1,\vec{y}_2,\omega_2)\cos\theta_{\vec{y}_2}\cdot\delta(\vec{y}_2 - \vec{x}_1)}{\Phi_1(1 + a_{\vec{x}_1}(\omega_1))}.$$

Sampling according to this mixed, discrete-continuous probability density can be realized in the following way. First it is decided randomly whether we sample L^e or the newly generated point using probabilities $1/(1+a_{\vec{x}_1}(\omega_1))$ and $a_{\vec{x}_1}(\omega_1)/(1 + a_{\vec{x}_1}(\omega_1))$, respectively. If L^e is selected, then the sampling process is the same as before, i.e. a random point and a random direction are found with probability density

$$\frac{L^e(\vec{y}_2,\omega_2)\cos\theta_{\vec{y}_2}}{\Phi_1}.$$

However, if the new point is chosen, then the direction of the next transfer is found with probability density

$$\frac{f_r(\omega_1,\vec{y}_2,\omega_2)\cos\theta_{\vec{y}_2}}{a_{\vec{x}_1}(\omega_1)}.$$

In either case, a ray defined by the selected point and direction is traced, and the complete power $\Phi_2 = \Phi_1 \cdot (1 + a_{\vec{x}_1}(\omega'_1))$ is transferred to that point which is hit by the ray. The subsequent steps of the iteration are similar.

Interestingly this iteration is a sequence of variable length random walks, since at each step the point that is last hit by the ray is only selected with a given probability as the starting point of the next ray.

To understand how it works, first let us assume that the environment is closed, that is, rays always hit objects. The algorithm selects a point from a lightsource and then starts a random walk. The algorithm computes a step by sending a ray to the randomly selected direction. At the first hit the image contribution is computed. To prepare for the next step and the power to be transferred is set to $(1+a_1)\Phi$. The walk finishes after the first step according to the powers, that is with probability $1/(1+a_1)$ and continues with probability $a_1/(1+a_1)$. If a walk finishes, another walk is initiated from the lightsource, if not, then the ray is emanated from the previous hit point. The new ray is traced, the image contribution is computed, and the power is set again to $(1+a_2(1+a_1))\Phi$. Thus now the walk is continued with

$$\frac{a_2+a_2a_1}{1+a_2+a_2a_1}$$

probability. Note that the limiting case of this continuation probability is a special average of the albedos:

$$\Pr\{\text{continue at step } n\} = \tilde{a} = \frac{a_n + a_na_{n-1} + a_na_{n-1}a_{n-2} + \ldots + a_n\cdot\ldots\cdot a_1}{1 + a_n + a_na_{n-1} + a_na_{n-1}a_{n-2} + \ldots + a_n\cdot\ldots\cdot a_1}$$

If all points have the same albedo a, then in the limiting case $\tilde{a}=a$, thus the continuation probability is similar to what is usually used in expansion. However, a difference still remains. When expansion continues a walk, it scales the power by $1/a_i$ according to Russian-roulette and resets the scaling factor to 1 when a new walk is initiated. In iteration, however, the power is scaled only by a_i, but the emission power is added at each reflection, and this is not reinitialized when a new lightsource point is sampled.

Now let us also consider open environments where a ray can leave the space with positive probability. When this happens, the power will be the emission power and the algorithm starts to build up the walk from scratch again. Generally, the continuation probability will be in between $\tilde{a}/(1+\tilde{a})$ and $\tilde{a}$ depending on how open the scene is.

Note that even this simple example highlighted the fundamental difference between iteration and expansion. Expansion — i.e. random walk — methods usually continue the walk with the probability of the local albedo, that is, they make decisions based on local characteristics. Iteration, on the other hand, takes into account the complete radiance function — it makes decision using global information — which eventually results in a continuation probability which depends on the average albedo and on how open the scene is. Since these factors determine the contraction of the light transport operator, we can state that iteration uses a more precise approximation of the contraction to randomly terminate the walk.

10.2.2 Stochastic iteration using ray-bundles

Let us recall that finite-element formulation needed by ray-bundles modifies the rendering equation to the following form:

$$\mathbf{L}(\omega) = \mathbf{L}^e(\omega) + \int_{\Omega} \mathbf{F}(\omega',\omega)\cdot\mathbf{A}(\omega')\cdot\mathbf{L}(\omega')\,d\omega'. \tag{10.15}$$

This section proposes a stochastic iteration type solution for the solution of this equation. Let us define a random operator $\mathcal{T}_F^*$ that behaves like the projected transport operator in the average case in the following way:

A random direction is selected using a uniform distribution and the radiance of all patches is transferred into this direction.

Formally, the definition is

$$\mathcal{T}^*(\omega')\mathbf{L}(\omega) = 4\pi\cdot\mathbf{T}(\omega',\omega)\cdot\mathbf{L}(\omega'). \tag{10.16}$$

If direction ω' is sampled from a uniform distribution — i.e. the probability density of the samples is $1/(4\pi)$ — then according to equation (8.17) the expected value of the application of this operator is

$$E[\mathcal{T}^*(\omega')\mathbf{L}(\omega)] = \int_{\Omega} 4\pi\cdot\mathbf{T}(\omega',\omega)\cdot\mathbf{L}(\omega')\,\frac{d\omega'}{4\pi} = \mathcal{T}_F\mathbf{L}(\omega). \tag{10.17}$$

In the definition of the random operator ω is the actually generated and ω' is the previously generated directions. Thus a "randomization point" is a global direction in this method.

The resulting algorithm is quite simple. In a step of the stochastic iteration an image estimate is computed by reflecting the previously computed radiance estimate towards the eye, and a new direction is found and this direction together with the previous direction are used to evaluate the random transport operator. Note that the algorithm requires just one variable for each patch i, the previous radiance $L[i]$.

The ray-bundle based stochastic iteration algorithm is:

```
Generate the first random global direction $\omega_1$
**for** each patch $i$ **do** $L[i] = L_i^e(\omega_1)$
**for** $m = 1$ **to** $M$ **do**                                              // *iteration cycles*
    Calculate the image estimate relfecting the incoming radiance $L[1], L[2], \ldots L[n]$ from $\omega_m$ towards the eye
    Average the estimate with the Image
    Generate random global direction $\omega_{m+1}$
    **for** each patch $i$ **do** $L^{\rm new}[i] = L_i^e(\omega_{m+1}) + 4\pi \cdot \sum_{j=1}^{n} \tilde{f}_i(\omega_m, \omega_{m+1}) \cdot A(i, j, \omega_m)/A_i \cdot L[j]$
**endfor**
Display Image
```

Using Phong interpolation, on the other hand, the radiance is evaluated at each point visible through a given pixel using the incoming radiance field, the surface normal and the BRDF of the found point.

Due to the fact that $A(i, j, \omega')$, $A(j, i, -\omega')$ can simultaneously be computed, the algorithm can easily be generalized to simultaneously handle bi-directional transfers.

Note that this algorithm is quite similar to the global walk algorithm, but it does not reinitialize the irradiance vector after each Dth step. In fact, it generates a single infinite walk, and adds the effect of the lightsources to the reflected light field and computes the image accumulation after each step.

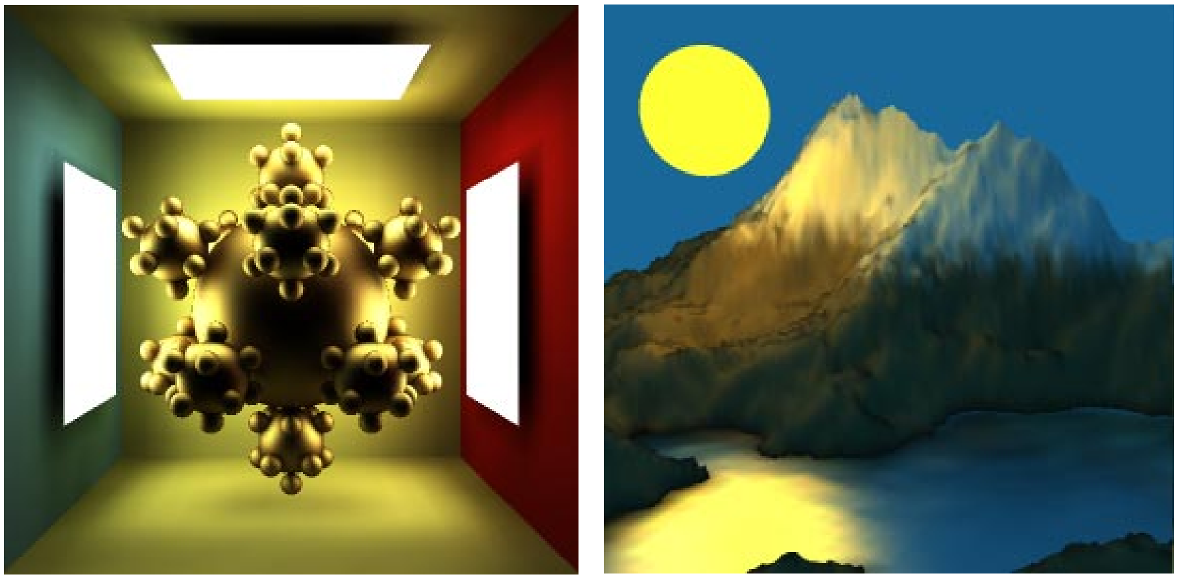


Figure 10.3: Images rendered by stochastic iteration

Chapter 11

Implementation of the path-tracing algorithm

This chapter presents a C++ program that implements the **path-tracing** algorithm.

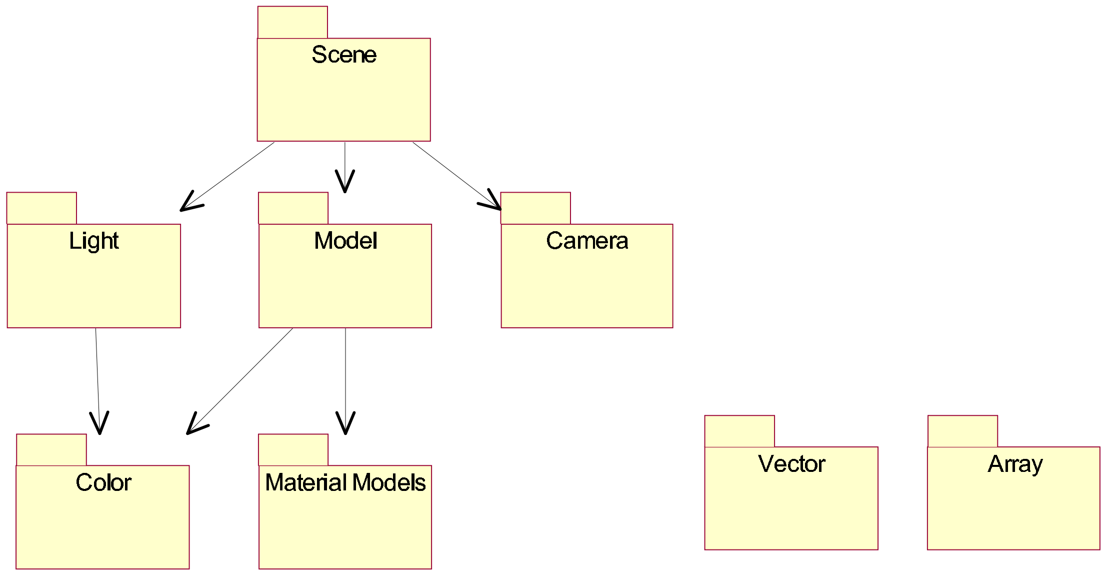


Figure 11.1: UML component diagram of the path tracer

A path-tracer consists of the following modules (figure 11.1):

- **Scene module** in which the path tracing is executed.
- **Model module** that stores the objects of the scene, which can be interrogated in geometric sense when tracing rays, and also optically when reflections are calculated.
- **Light module** that introduces emission functions and abstract lightsources in the scene.
- **Camera module** that represents the camera parameters.
- **Material models module** participates in the specification of objects by defining BRDFs, albedos and functions for BRDF sampling.
- **Color module** can handle spectra and color information.
- **Array module** is a supplementary module that provides a generic, dynamic array.
- **Vector module** is also a supplementary module that defines 2D and 3D vector classes and implements all vector operations.

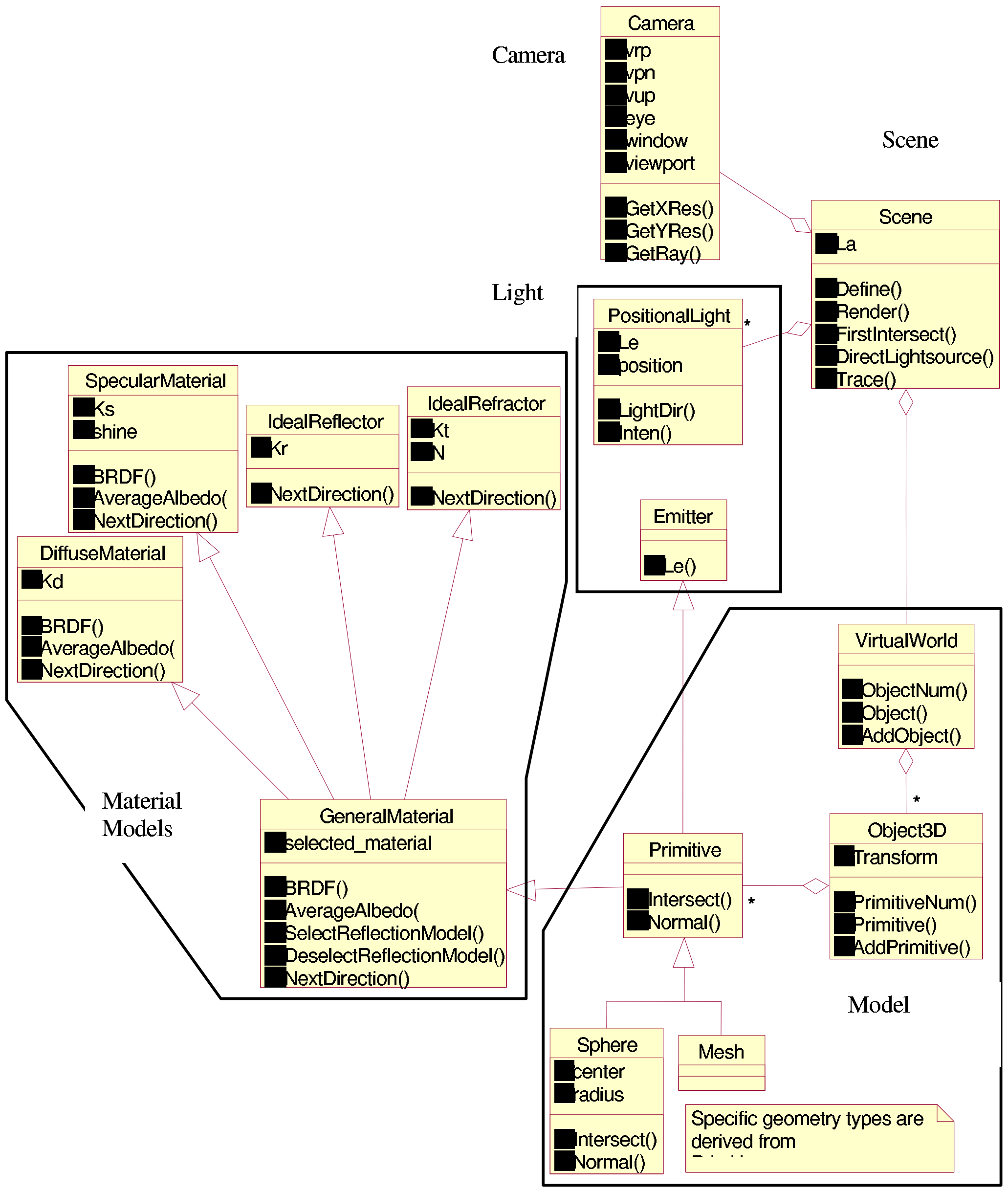


Figure 11.2: UML class diagram of the path tracer

11.1 Vector module

11.1.1 Point3D class

The `Point3D` class defines a 3D point and implements all operations, including addition, subtraction, scalar and vector products, multiplication and division with a scalar, length computation and normalization. Type `Vector3D` is equivalent to `Point3D`.

```
//===============================================================
class Point3D {         // 3D point in Cartesian coordinates
//===============================================================
    Coord x, y, z;
public:
    Point3D( Coord x0 = 0, Coord y0 = 0, Coord z0 = 0 ) { x = x0; y = y0; z = z0; }
    Point3D operator-( ) { return Point3D( -x, -y, -z ); }
    Point3D operator+( Point3D& p ) { return Point3D( x + p.x, y + p.y, z + p.z ); }
    Point3D operator-( Point3D& p ) { return Point3D( x - p.x, y - p.y, z - p.z ); }
    void    operator+=( Point3D& p ) { x += p.x; y += p.y; z += p.z; }
    void    operator/=( double d ) { x /= d; y /= d; z /= d; }
    Point3D operator*( double s ) { return Point3D( x * s, y * s, z * s ); }
    Point3D operator/( double s ) { return Point3D( x / s, y / s, z / s ); }
    Point3D operator%( Point3D& v ) {                                    // vector product
        return Point3D(y * v.z - z * v.y, z * v.x - x * v.z, x * v.y - y * v.x);
    }
    double  operator*( Point3D& p ) { return (x * p.x + y * p.y + z * p.z); }
    double  Length( ) { return sqrt( x * x + y * y + z * z); }
    void    Normalize() { *this = (*this) * (1/Length()); }

    Coord& X() { return x; }
    Coord& Y() { return y; }
    Coord& Z() { return z; }
};

typedef Point3D Vector3D;
```

11.1.2 Transformation class

Class `Transform3D` represents a homogeneous linear transformation, which can be built according to different transformation types, can be concatenated and can be applied to 3D points.

```
//===============================================================
class Transform3D {             // 3D homogeneous linear transformation
//===============================================================
    double m[4][4];             // 4 x 4 transformation matrix
public:
    Transform3D( );
    Transform3D( Point3D v1, Point3D v2 , Point3D v3, Point3D v4 );
    Point3D Transform( Point3D& p );
};
```

11.2 Container module

The `Array` is a container class that implements a generic, dynamic array. Dynamic array means that the maximum number of elements is not fixed, but the array may reallocate itself if an index refers to an element that has not been defined yet.

```
//-------------------------------------------------------------------
template < class Type >  class Array {
//-------------------------------------------------------------------
        int    alloc_size;             // allocated number of elements
        int    size;                   // maximum index of used elements
        Type * array;                  // start address on the heap
public:
        Array( int s = 0 );
        Type& operator[] ( int idx );
        int   Size( ) { return size; }
};
```

11.3 Color module

Colors are defined by the `SColor` class. Objects of this class are spectrum functions whose values are stored in array `I` on wavelengths stored in array `lambdas`. The `ColorMatch` member function computes the equivalent R, G, B primaries. `SColor` also provides a set of operations on spectra, including addition, subtraction, multiplication and division by a scalar, etc.

```
//===============================================================
class SColor {  // spectrum
//===============================================================
    static double lambdas[NLAMBDA];        // wavelengths
    double I[NLAMBDA];                     // intensities

    void ColorMatch( double lambda, double& r, double& g, double& b );
public:
    SColor( ) { for(int l = 0; l < NLAMBDA; l++) I[l] = 0; }
    SColor( double gray ) { for(int l = 0; l < NLAMBDA; l++) I[l] = gray; }

    double& Int( int l ) { return I[l]; }

    SColor operator*( double s ) {
        SColor res; for(int l = 0; l < NLAMBDA; l++) res.I[l] = I[l] * s; return res;
    }
    SColor operator*( SColor& m ) {
        SColor res; for(int l = 0; l < NLAMBDA; l++) res.I[l] = I[l] * m.I[l]; return res;
    }
    void  operator*=( SColor& m ) { (*this) = (*this) * m; }
    SColor operator/( double d ) {
        SColor res; for(int l = 0; l < NLAMBDA; l++) res.I[l] = I[l] / d; return res;
    }
    void  operator/=( double d ) { (*this) = (*this) / d; }
    SColor operator+( SColor& d ) {
        SColor res; for(int l = 0; l < NLAMBDA; l++) res.I[l] = I[l] + d.I[l]; return res;
    }
    void  operator+=( SColor& d ) { (*this) = (*this) + d; }
    double Luminance( ) {
        double sum = 0; for(int l = 0; l < NLAMBDA; l++) sum += I[l] / NLAMBDA; return sum;
    }

    int operator!=( double c ) { return ( c != Luminance( ) ); }

    double Red();
    double Green();
    double Blue();
};
```

11.4 Material models

The optical response to a point lightsource and to spherical illumination are defined by a BRDF function and an albedo, respectively. The return value of a `BRDF` is a spectrum since the BRDF can depend on the wavelength. In path tracing the albedo is used to randomly terminate the walk. Since the albedo is also wavelength dependent, its luminance value is used as a probability of continuing the walk. To realize importance sampling, material models also have a `NextDirection` member function which generates a random direction `L` with a probability that is approximately proportional to the BRDF times the cosine of the angle between the surface normal and the generated direction. The `UNIFORM(i)` macro provides a pseudo-random or quasi-random value uniformly distributed in $[0, 1]$. Its input parameter is relevant only for quasi-Monte Carlo quadrature, when `UNIFORM(i)` generates the next sample in the ith independent low-discrepancy series. Note that when looking at the Neumann series as a high-dimensional integral, the direction of each next bounce corresponds to two additional integration variables and Russian-roulette also uses a new variable at each bounce. Low-discrepancy series number 0 and 1 are used for sampling the pixel area. Series number 2 is involved in the decision whether the first bounce is to be computed by Russian-roulette. Series number 3 and 4 are used to determine the direction of the first bounce, etc.

11.4.1 Diffuse material class

Diffuse materials reflect light uniformly into all directions. The `BRDF` of the `DiffuseMaterial` class is thus constant. For importance sampling, function `NextDirection` generates a random direction in `L` with a probability density that is proportional to the cosine of the outgoing angle. The function also returns the probability of the generated sample which will be used to divide the contribution of the selected direction, as required by importance sampling. Note that this function also gets parameter `d` that selects which low-discrepancy series should be sampled. This parameter is equal to the index of the bounce actually computed.

```
//===============================================================
class DiffuseMaterial {
//===============================================================
    SColor Kd;
public:
    SColor& kd() { return Kd; }
    SColor BRDF(Vector3D& L, Vector3D& N, Vector3D& V) { return Kd; }

    double NextDirection(Vector3D& L, Vector3D& N, Vector3D& V, int d) {
        double u = UNIFORM(3*d + 3), v = UNIFORM(3*d + 4);

        double theta = asin(sqrt(u)), phi = M_PI * 2.0 * v;
        Vector3D O = N % Vector3D(0, 0, 1);
        if (O.Length() < EPSILON) O = N % Vector3D(0, 1, 0);
        Vector3D P = N % O;
        L = N * cos(theta) + O * sin(theta) * cos(phi) + P * sin(theta) * sin(phi);
        double prob = cos(theta) / M_PI;
        return prob;
    }
    double AverageAlbedo( Vector3D& N, Vector3D& V ) { return Kd.Luminance() * M_PI; }
};
```

11.4.2 Ideal mirror class

An **ideal mirror** reflects light only to the ideal reflection direction. Its BRDF is thus infinite at a single direction and zero in all other directions. Such a Dirac-delta function cannot be represented directly, thus this class does not have BRDF member function. Instead, we can determine the direction in which reflection can happen by function `NextDirection`.

```
//===============================================================
class IdealReflector {
//===============================================================
public:
    SColor& kr(Vector3D& N, Vector3D& V);

    double NextDirection(Vector3D& L, Vector3D& N, Vector3D& V) {
        L = N * (N * V) * 2 - V; return 1;
    }
    double AverageAlbedo( Vector3D& N, Vector3D& V ) { return kr(N, V).Luminance(); }
};
```

Function `kr` may compute the Fresnel function on different wavelengths. For plastics, `kr` is usually assumed to be constant. For metals, however, the wavelength and incident angle dependency defined by the Fresnel function is not negligible. The following program provides the complex refraction index of aluminum and gold, and computes the values of the Fresnel function.

```
#define NREFIDX 8

struct RefIdx {double lambda, n, k; }
    alu[NREFIDX] = {
        {400, 0.4900, 4.8600},
        {450, 0.6180, 5.4700},
        {500, 0.7690, 6.0800},
        {550, 0.9580, 6.6900},
        {600, 1.2000, 7.2600},
        {650, 1.4700, 7.7900},
        {700, 1.8300, 8.3100},
        {750, 2.4000, 8.6200}
    },
    gold[NREFIDX] = {
        {400, 1.6580, 1.9560},
        {450, 1.5102, 1.8788},
        {500, 0.8469, 1.8753},
        {550, 0.3485, 2.7144},
        {600, 0.2177, 2.9097},
        {650, 0.1676, 3.1138},
        {700, 0.1605, 3.9784},
        {750, 0.1680, 4.5886},
    };

//===============================================================
class FresnelFunction {
//===============================================================
    Array<RefIdx> refidx;
public:
    FresnelFunction( RefIdx r[] ) { for(int l = 0; l < NREFIDX; l++) refidx[l] = r[l]; }
    double Fresnel( double lambda, double theta ) {
        double n, k;
        for( int l = 1; l < refidx.Size(); l++ ) {       // linear interpolation for n,k
            if (lambda < refidx[l].lambda) {
                double la2 = lambda - refidx[l-1].lambda;
                double la1 = refidx[l].lambda - lambda;
                double la = la1 + la2;
                n = (la1 * refidx[l-1].n + la2 * refidx[l].n)/la;
                k = (la1 * refidx[l-1].k + la2 * refidx[l].k)/la;
                break;
            }
        }
        double t1 = n*n - k*k - sin(theta) * sin(theta);
        double t2 = sqrt(t1*t1 + 4.0*n*n*k*k);
        double a2 = 0.5 * (t2 + t1), a = sqrt(a2);
        double t3 = a2 + 0.5 * (t2 - t1);
        double t4 = 2.0 * a * cos(theta);
        double fsd = (t3 + t4 + cos(theta) *cos(theta));
        double Fs = (fsd > EPSILON) ? (t3 - t4 + cos(theta) *cos(theta))/fsd : 0;
        double t5 = 2.0 * a * sin(theta) * tan(theta);
        double t6 = t3 + sin(theta)*sin(theta) * tan(theta)*tan(theta);
        double Fp = (t6 + t5 > EPSILON) ? Fs * (t6 - t5)/(t6 + t5) : 0;
        return ((Fp + Fs)/2.0);
    }
};
```

11.4.3 Ideal refracting material class

An ideal refracting material can also refract into a single direction which is computed by `NextDirection` from the refraction index of the material (`N`). Input parameters also include variable `out` that indicates whether the surface has been approached from outside the material or from inside the material. If we come from inside, then we should use reciprocal of the refraction index during the computation. The return value of the function shows whether or not there is refraction direction due to **total reflection**.

```
//===============================================================
class IdealRefractor {
//===============================================================
    SColor  Kt;
    double N;
public:
    IdealRefractor( ) : Kt(0) { N = 1; }
    SColor& kt( ) { return Kt; }
    double& n( ) { return N; }
    double NextDirection(Vector3D& L, Vector3D& N, Vector3D& V, BOOL out) {
        double cn = ( out ) ? n( ) : 1.0/n( );
        double cosa = N * V;
        double disc = 1 - (1 - cosa * cosa) / cn / cn;
        if (disc < 0) return 0;
        L = N * (cosa / cn - sqrt(disc)) - V / cn;
        return 1;
    }
};
```

11.4.4 Specular material class

The `SpecularMaterial` class simulates the reciprocal Phong type reflection. The `Ks` reflection coefficient is the approximation of the albedo. As for ideal mirrors `Ks` can be supposed to be constant or set proportionally to the Fresnel function. The `shine` parameter describes how smooth the surface is.

```
//=============================================================
class SpecularMaterial {
//=============================================================
    SColor  Ks;
    double shine;
public:
    SpecularMaterial( ) : Ks(0) { shine = 10; }
    SColor& ks() { return Ks; }
    double& Shine( ) { return shine; }
    SColor BRDF(Vector3D& L, Vector3D& N, Vector3D& V);

    double NextDirection(Vector3D& L, Vector3D& N, Vector3D& V, double d) {
        double u = UNIFORM(3*d + 3), v = UNIFORM(3*d + 4);
        double cos_ang_V_R = pow(u, 1.0/(shine+1) );
        double sin_ang_V_R = sqrt( 1.0 - cos_ang_V_R * cos_ang_V_R );

        Vector3D O = V % Vector3D(0, 0, 1);
        if (O.Length() < EPSILON) O = V % Vector3D(0, 1, 0);
        Vector3D P = O % V;
        Vector3D R = O * sin_ang_V_R * cos(2.0 * M_PI * v) +
                     P * sin_ang_V_R * sin(2.0 * M_PI * v) +
                     V * cos_ang_V_R;
        L = N * (N * R) * 2.0 - R;
        double cos_ang_N_L = N * L;
        if (cos_ang_N_L < 0) return 0;
        double prob = (shine+1)/2/M_PI * pow(cos_ang_V_R, shine);
        return prob;
    }
    double AverageAlbedo( Vector3D& N, Vector3D& V ) { return ks().Luminance() * (N * V); }
};
```

11.4.5 General material class

A general material may reflect some portion of the incoming light diffusely, some other portion specularly or as an ideal mirror, and may also refract some portion according to the law of refraction. Thus a general material combines the properties of all elementary material models.

BRDF sampling select from the elementary BRDFs randomly, according to the average albedos. Member function `SelectReflectionModel` makes this random selection and stores the result in the `selected_material` variable, or it may decide that the random walk should be terminated according to Russian-roulette. The calculation of the next random direction and the BRDF is based on only the selected model. Since in the program direct lightsource calculation handles only point lightsources and the probability that a random direction finds a point lightsource is zero, direct lightsource calculation uses only diffuse and specular components. To prepare for this, variable `selected_material` is reset by the `DeselectReflectionModel` function.

```
//=============================================================
class GeneralMaterial : public DiffuseMaterial, public SpecularMaterial,
                        public IdealReflector, public IdealRefractor {
//=============================================================
    enum {NO, DIFFUSE, SPECULAR, REFLECTOR, REFRACTOR, INCOHERENT} selected_material;
public:
    GeneralMaterial( ) { selected_material = INCOHERENT; }

    double SelectReflectionModel(Vector3D& N, Vector3D& V, int d) {
        double akd = DiffuseMaterial :: AverageAlbedo(N, V);
        double aks = SpecularMaterial :: AverageAlbedo(N, V);
        double akr = kr(N, V).Luminance();
        double akt = kt().Luminance();
        double r = UNIFORM(3*d + 2);
        if ((r -= akd) < 0) { selected_material = DIFFUSE;  return akd; }
        if ((r -= aks) < 0) { selected_material = SPECULAR; return aks; }
        if ((r -= akr) < 0) { selected_material = REFLECTOR; return akr; }
        if ((r -= akt) < 0) { selected_material = REFRACTOR; return akt; }
        selected = NO; return 0.0;            // russian roulette
    }
    double NextDirection(Vector3D& L, Vector3D& N, Vector3D& V, BOOL out, int d) {
        switch (selected_material) {
        case DIFFUSE:   return DiffuseMaterial :: NextDirection(L, N, V, d);
        case SPECULAR:  return SpecularMaterial :: NextDirection(L, N, V, d);
        case REFLECTOR: return IdealReflector :: NextDirection(L, N, V);
        case REFRACTOR: return IdealRefractor :: NextDirection(L, N, V, out);
        default:        return 0;
        }
    }
    void DeselectReflectionModel( ) { selected_material = INCOHERENT; }
    SColor BRDF(Vector3D& L, Vector3D& N, Vector3D& V) {
        double cost;
        switch (selected_material) {
        case DIFFUSE:   return DiffuseMaterial :: BRDF(L, N, V);
        case SPECULAR:  return SpecularMaterial :: BRDF(L, N, V);
        case REFLECTOR: cost = N * L;
                        if (cost > EPSILON) return (kr(N, V) / cost);
                        else                return SColor(0);
        case REFRACTOR: cost = -(N * L);
                        if (cost > EPSILON) return (kt( ) / cost);
                        else                return SColor(0);
        case ALL:       return (DiffuseMaterial :: BRDF(L, N, V) +
                               SpecularMaterial :: BRDF(L, N, V));
        default:        return SColor(0);
        }
    }
};
```

11.5 Light module

In addition to the background light, illumination can be introduced in two ways. Either primitives may have non-zero emission, or abstract, point-lightsources may be placed at certain locations.

11.5.1 Emitter class

An `Emitter` emits light:

```
//=============================================================
class Emitter  {
//=============================================================
    SColor LE;
public:
    SColor& Le() { return LE; }
};
```

11.5.2 Positional light class

`PositionalLight` represents abstract, point-lightsources:

```
//=============================================================
class PositionalLight {
//=============================================================
    Point3D pos;
    SColor  intensity;
public:
    PositionalLight( Point3D& p, SColor& inten ) : intensity(inten), pos(p) { }
    Point3D& Pos( ) { return pos; }
    Vector3D LightDir(Point3D& x) { return (pos - x).Normalize(); }
    SColor Le(Point3D& x, Vector3D& dir) { return (intensity / ((x-pos)*(x-pos)*4*M_PI)); }
};
```

11.6 Model module

A virtual world model is hierarchical. A model is built of objects which, in turn, consist of primitives.

11.6.1 Primitive class

A `Primitive3D` is defined by its material properties and by its geometry. The geometry can be interrogated by two functions: `Intersect` answers whether or not a ray intersects the object; `Normal` provides the normal vector at a surface point.

```
//===============================================================
class Primitive3D : public Emitter, public GeneralMaterial {
//===============================================================
public:
    virtual double Intersect( Ray& r ) = 0;
    virtual Vector3D Normal( Point3D& x ) = 0;
};
```

In order to implement the geometric queries, the shape of the object must be known. For example, a `Sphere` class can be defined in the following way:

```
//===============================================================
class Sphere : public Primitive3D {
//===============================================================
    Point3D center;
    double  radius;
public:
    Sphere(Point3D& cent, double rad) { center = cent; radius = rad; }

    Vector3D Normal(Point3D& x) { return ((x - center)/radius); }

    double Intersect( Ray& r ) {
        Vector3D dist = r.Start() - center;
        double b = (dist * r.Dir()) * 2.0;
        double a = (r.Dir() * r.Dir());
        double c = (dist * dist) - radius * radius;

        double discr = b * b - 4.0 * a * c;
        if ( discr < 0 ) return -1;
        double sqrt_discr = sqrt( discr );
        double t1 = (-b + sqrt_discr)/2.0/a;
        double t2 = (-b - sqrt_discr)/2.0/a;
        if (t1 < EPSILON) t1 = -EPSILON;
        if (t2 < EPSILON) t2 = -EPSILON;
        if (t1 < 0 && t2 < 0) return -1;

        double t;
        if ( t1 < 0 && t2 >= 0 )        t = t2;
        else if ( t2 < 0 && t1 >= 0 )   t = t1;
        else if (t1 < t2)               t = t1;
        else                            t = t2;
        return t;
    }
};
```

11.6.2 Object class

An `Object3D` is a collection of primitives and a transform that moves the primitives from the local modeling coordinate system to the world coordinate system.

```
//===============================================================
class Object3D {
//===============================================================
    Array<Primitive3D *> prs;
    Transform3D          tr;
public:
    void AddPrimitive( Primitive3D * p ) { prs[ prs.Size() ] = p; }
    Primitive3D * Primitive( int i ) { return prs[i]; }
    Transform3D& Transform( ) { return tr; }
    int PrimitiveNum() { return prs.Size(); }
};
```

11.6.3 Virtual world class

The virtual world is a collection of objects.

```
//===============================================================
class VirtualWorld {
//===============================================================
    Array<Object3D *>  objs;
public:
    Object3D * Object( int o ) { return objs[o]; }
    void AddObject( Object3D * o ) { objs[ objs.Size() ] = o; }
    int ObjectNum() { return objs.Size(); }
};
```

11.7 Camera module

A `Camera3D` is a collection of parameters that define a generally positioned and oriented 3D camera with general zoom settings.

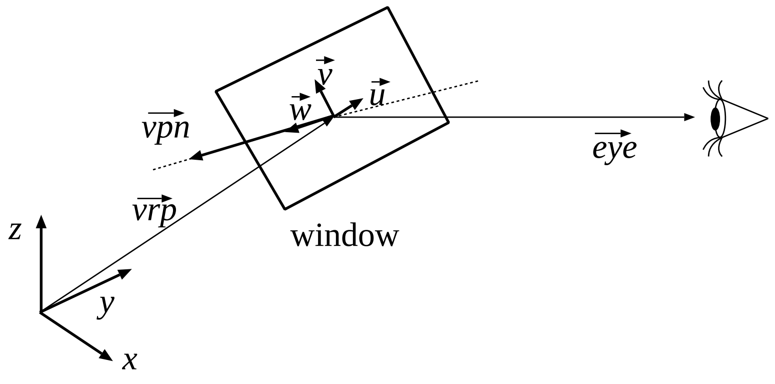


Figure 11.3: Parameters of the camera

The `GetRay` function provides a ray which starts at the eye and goes through the specified point on the window.

```
//===============================================================
class Camera3D {
//===============================================================
protected:
    Point3D vrp;
    Vector3D vpn, vup, eye,u, v, w, world_eye;
    RectAngle window, viewport;

    void CalcViewTranform( ) {
        w = vpn; w.Normalize( );
        u = w % vup; u.Normalize( );
        v = u % w;
        Transform3D Tuvw( u, v, w, vrp );
        world_eye = Tuvw.Transform( eye );
    }
public:
    void SetCamera( Point3D& vrp0, Vector3D& vpn0, Vector3D& vup0,
                    Vector3D& eye0, RectAngle& window0) {
        vrp = vrp0; vpn = vpn0; vup = vup0; eye = eye0; window = window0;
    }
    RectAngle Viewport( ) { return viewport; }
    void SetViewport( RectAngle v ) { viewport = v; }
    Ray GetRay( Coord X, Coord Y ) {
        double x, y;
        x = window.HSize()/viewport.HSize() * (X - viewport.HCenter());
        y = window.VSize()/viewport.VSize() * (Y - viewport.VCenter());
        Vector3D dir = u * x + v * y ;
        return Ray( world_eye + vrp, dir );
    }
};
```

11.8 Scene module

11.8.1 Scene class

The scene consists of a virtual world model, a camera, a number of abstract lightsources and also the background illumination (i.e. sky-light).

```
//=============================================================
class Scene {
//=============================================================
    VirtualWorld              world;            // virtual world model
    Camera3D                  camera;           // camera
    Array< PositionalLight * > lightsources;     // abstract, point-lightsources
    SColor                    La;               // background illumination

    Primitive3D * Intersect( Ray& r, Point3D& x );
    BOOL        IntersectShadow( Ray r, double maxt );
    SColor      DirectLightsource(Primitive3D * q, Vector3D& V, Vector3D& N, Point3D& x);
    SColor      Trace(Ray r, int depth);
public:
    void Define( );
    void Render( );
};
```

11.9 Dynamic model of path tracing

The main controller of the program is the `Scene` class. The operation is shown in the sequence diagram of figure 11.4.

11.9.1 Finding the visible primitive

The `Intersect` member function finds that primitive which is first intersected by the ray, and returns both the primitive and the location of the intersection (`x`).

```
//-------------------------------------------------------------
Primitive3D * Scene :: Intersect( Ray& r, Point3D& x ) {
//-------------------------------------------------------------
    double    t = -1;
    Primitive3D * po = NULL;
    for(int actobj = 0; actobj < world.ObjectNum(); actobj++) {
        for(int actprim = 0; actprim < world.Object(actobj)->PrimitiveNum(); actprim++) {
            Primitive3D * p = world.Object(actobj)->Primitive(actprim);
            double tnew = p -> Intersect( r );
            if ( tnew > 0 && (tnew < t || t < 0) ) { t = tnew; po = p; }
        }
    }
    if (t > 0) x = r.Start() + r.Dir() * t;
    return po;
}
```

11.9.2 Detecting the visible lightsources

The `IntersectShadow` function checks whether or not a point lightsource is visible from a point. It traces the shadow rays until the maximal ray-parameter `maxt` corresponding to the lightsource, and checks whether any object is intersected before reaching the lightsource.

```
//-------------------------------------------------------------
BOOL Scene :: IntersectShadow( Ray r, double maxt ) {
//-------------------------------------------------------------
    for(int actobj = 0; actobj < world.ObjectNum(); actobj++) {
        for(int actprim = 0; actprim < world.Object(actobj)->PrimitiveNum(); actprim++) {
            Primitive3D * p = world.Object(actobj)->Primitive(actprim);
            double t = p -> Intersect( r );
            if ( t > 0 && t < maxt ) return TRUE;
        }
    }
    return FALSE;
}
```

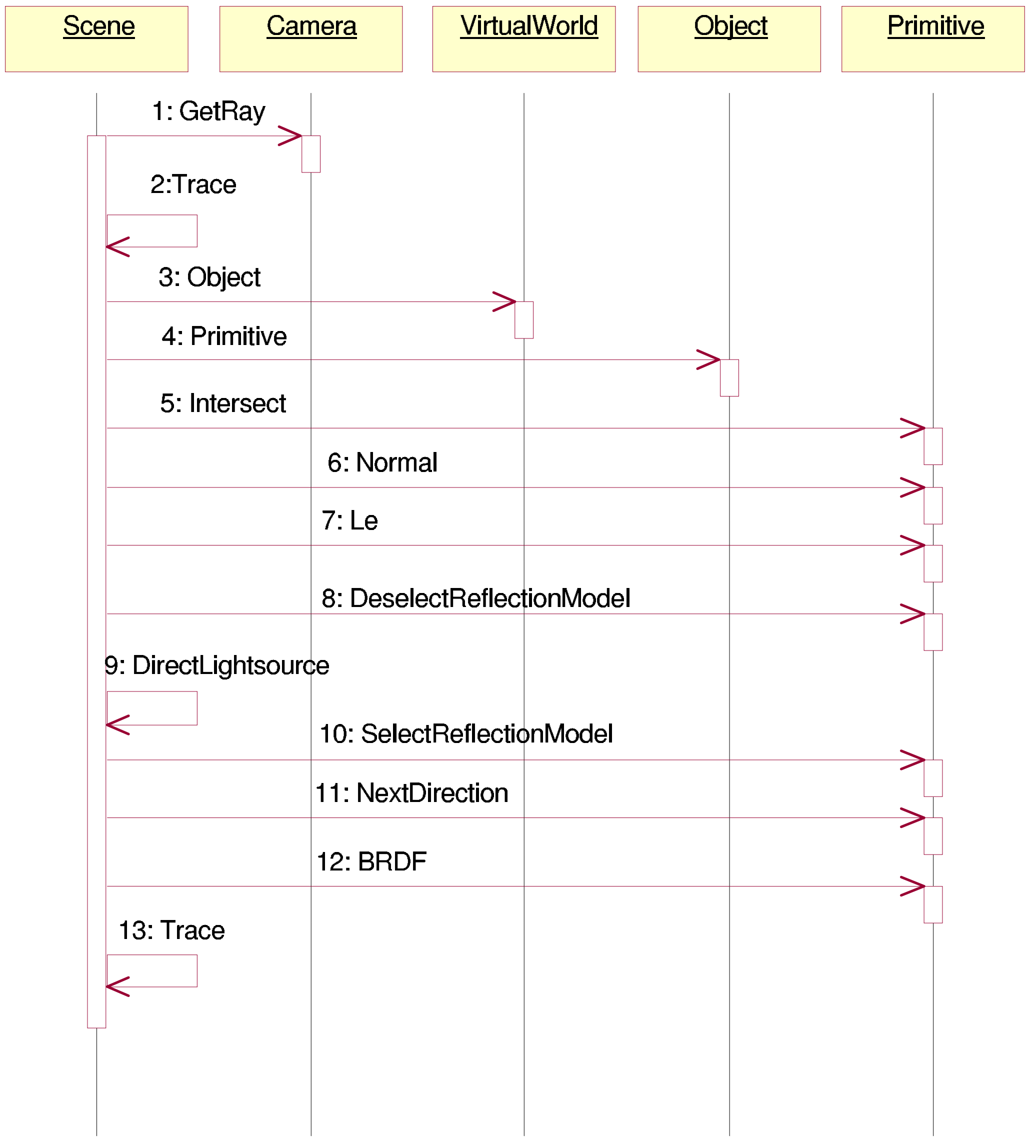


Figure 11.4: Sequence diagram of path tracer

11.9.3 Direct lightsource computation

Member function `DirectLightsource` determines the single reflection of point lightsources.

```
//-----------------------------------------------------------------
SColor Scene :: DirectLightsource(Primitive3D * q, Vector3D& V, Vector3D& N, Point3D& x) {
//-----------------------------------------------------------------
    q->DeselectReflectionModel( );
    SColor c;
    for(int l = 0; l < lightsources.Size(); l++) {
        Vector3D L = lightsources[l] -> Pos() - x;
        double lightdist = L.Length();
        L /= lightdist;
        if ( IntersectShadow(Ray(x, L), lightdist) == FALSE ) {    // is lightsource occluded?
            double cost = N * L;
            if (cost > 0) c += q->BRDF(L, N, V) * lightsources[l] -> Le(x, -L) * cost;
        }
    }
    return c;
}
```

11.9.4 Path tracing

The core of the program is the `Trace` function which recursively traces rays. On a single level of the recursion a ray passed as a parameter is traced and the point hit by the ray is identified. The sum of the emission and the reflection of the direct lightsources is computed. In order to continue the walk, first an elementary BRDF model is selected according to the average albedos or it is decided that the walk is stopped according to Russian roulette. If the walk is to be continued, then a random next direction is found using BRDF sampling. The radiance coming from this new direction is computed by the recursive call of the same function. The incoming radiance is multiplied by the BRDF and the cosine of the incoming angle and divided by the selection probability as required by importance sampling.

```
//-----------------------------------------------------------------
SColor Scene :: Trace(Ray r, int d) {
//-----------------------------------------------------------------
    if (d > MAXDEPTH) return La;
    Point3D x;
    Primitive3D * q = Intersect(r, x);
    if (q == NULL) return La;

    Vector3D normal = q -> Normal(x);
    BOOL out = TRUE;
    if ( normal * (-r.Dir()) < 0) { normal = -normal; out = FALSE; }

    SColor c = q -> Le(x, -r.Dir());
    c += DirectLightsource(q, -r.Dir(), normal, x);

    double prob = q->SelectReflectionModel( normal, -r.Dir(), d );
    if (prob < EPSILON) return c;                  // Russian roulette

    Vector3D newdir;
    prob *= q->NextDirection( newdir, normal, -r.Dir(), out, d );
    if (prob < EPSILON) return c;

    double cost = newdir * normal;
    if (cost < 0) cost = -cost;
    if (cost > EPSILON) {
        SColor w = q->BRDF( newdir, normal, -r.Dir() ) * cost;
        if (w.Luminance() > EPSILON) c += PathTrace( Ray(x, newdir), d+1 ) * w / prob;
    }
    return c;
}
```

Note that the `Trace` function also gets the level of recursion as a second parameter. On the one hand, this can be used to deterministically truncate the Neumann series. On the other hand, this depth parameter selects which low-discrepancy series should be used to find the next sample of the integral quadrature.

11.9.5 Rendering complete images

Rendering sends a number of rays through each pixel, gets their radiances and sets the pixel color according to their averages.

```
//-------------------------------------------------------------
void Scene :: Render( ) {
//-------------------------------------------------------------
    for(int y = 0; y < camera.Viewport().VSize(); y++) { // for each pixel of the viewport
        for(int x = 0; x < camera.Viewport().HSize(); x++) {
            SColor col(0);
            for( int i = 0; i < NSAMPLE; i++ ) {       // NSAMPLE samples are computed
                double dx = UNIFORM(0) - 0.5;          // uniform distribution on the pixel
                double dy = UNIFORM(1) - 0.5;
                Ray r = camera.GetRay(x + dx, y + dy); // Calculate the ray
                col += Trace( r, 0 );                  // Calculate the power of the ray
            }
            col /= NSAMPLE;                            // average
            WritePixel( x, y, col );
        }
    }
}
```

BIBLIOGRAPHY

[Ábr97] Gy. Ábrahám. *Optika*. Panem-McGraw-Hill, Budapest, 1997.

[AH93] L. Aupperle and P. Hanrahan. A hierarchical illumination algorithms for surfaces with glossy reflection. *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 155–162, 1993.

[AK90] J. Arvo and D. Kirk. Particle transport and image synthesis. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 63–66, 1990.

[Arv95] J. Arvo. Stratified sampling of spherical triangles. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 437–438, 1995.

[BBS96] G. Baranoski, R. Bramley, and P Shirley. Fast radiosity solutions for environments with high average reflectance. In *Rendering Techniques '96*, pages 345–355, 1996.

[Bek97] P. Bekaert. Error control for radiosity. In *Rendering Techniques '97 (8th Eurographics Workshop on Rendering)*, Porto, Portugal, 1997.

[BF89] C. Buckalew and D. Fussell. Illumination networks: Fast realistic rendering with general reflectance functions. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):89–98, July 1989.

[BKP91] J. C. Beran-Koehn and M. J. Pavicic. A cubic tetrahedral adaptation of the hemicube algorithm. In James Arvo, editor, *Graphics Gems II*, pages 299–302. Academic Press, Boston, 1991.

[Bli77] J. F. Blinn. Models of light reflection for computer synthesized pictures. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, pages 192–198, 1977.

[BNN$^+$98] P. Bekaert, L. Neumann, A. Neumann, M. Sbert, and Y. Willems. Hierarchical Monte-Carlo radiosity. In *Rendering Techniques '98*, pages 259–268, 1998.

[BS63] P. Beckmann and A. Spizzichino. *The Scattering of Electromagnetic Waves from Rough Surfaces*. MacMillan, 1963.

[BS95] P. Bodrogi and J. Schanda. Testing the calibration model of colour crt monitors. *Displays*, 16(3):123–133, 1995.

[CG85] M. Cohen and D. Greenberg. The hemi-cube, a radiosity solution for complex environments. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, pages 31–40, 1985.

[Chi88] H. Chiyokura. *Solid Modelling with DESIGNBASE*. Addision Wesley, 1988.

[CLSS97] P. H. Christensen, D. Lischinski, E. J. Stollnitz, and D. H. Salesin. Clustering for glossy global illumination. *ACM Transactions on Graphics*, 16(1):3–33, 1997.

[CMS98] F. Castro, R. Martinez, and M. Sbert. Quasi Monte-Carlo and extended first-shot improvements to the multi-path method. In *Spring Conference on Computer Graphics '98*, pages 91–102, 1998.

[CPC84] R. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, pages 137–145, 1984.

[CSK98] B. Csébfalvi and L. Szirmay-Kalos. Interactive volume rotation. *Machine Graphics and Vision*, 7(4):793–806, 1998.

[CSSD96] P. H. Christensen, E. J. Stollnitz, D. H. Salesin, and T. D. DeRose. Global illumination of glossy environments using wavelets and importance. *ACM Transactions on Graphics*, 15(1):37–71, 1996.

[CT81] R. Cook and K. Torrance. A reflectance model for computer graphics. *Computer Graphics*, 15(3), 1981.

[Dav54] H. Davis. The reflection of electromagnetic waves from a rough surface. In *Proceedings of the Institution of Electrical Engineers*, volume 101, pages 209–214, 1954.

[DBW97] Ph. Dutre, Ph. Bekaert, and Y. D. Willems. Bidirectional radiosity. In *Rendering Techniques '97*, pages 205–216, 1997.

[Deá89] I. Deák. *Random Number Generators and Simulation*. Akadémia Kiadó, Budapest, 1989.

[DLW93] Ph. Dutre, E. Lafortune, and Y. D. Willems. Monte Carlo light tracing with direct computation of pixel intensities. In *Compugraphics '93*, pages 128–137, Alvor, 1993.

[DW96] Ph. Dutre and Y. D. Willems. Potential-driven Monte Carlo particle tracing for diffuse environments with adaptive probability functions. In *Rendering Techniques '96*, pages 306–315, 1996.

[Erm75] S.M. Ermakow. *Die Monte-Carlo-Methode und verwandte Fragen.* R. Oldenbourg Verlag, Wien, 1975.

[FP94] M. Feda and W. Purgathofer. A median cut algorithm for efficient sampling of radiosity functions. In *Eurographics'94*, 1994.

[Gla95] A. Glassner. *Principles of Digital Image Synthesis.* Morgan Kaufmann Publishers, Inc., San Francisco, 1995.

[Hec91] P. S. Heckbert. *Simulating Global Illumination Using Adaptive Meshing.* PhD thesis, University of California, Berkeley, 1991.

[Her91] Ivan Herman. *The Use of Projective Geometry in Computer Graphics.* Springer-Verlag, Berlin, 1991.

[HMF98] M. Hyben, I. Martisovits, and A. Ferko. Scene complexity for rendering in flatland. In L. Szirmay-Kalos, editor, *Spring Conference on Computer Graphics*, pages 112–120, 1998.

[HSA91] P. Hanrahan, D. Salzman, and L. Aupperle. Rapid hierachical radiosity algorithm. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 1991.

[HTSG91] X. He, K. Torrance, F. Sillion, and D. Greenberg. A comprehensive physical model for light reflection. *Computer Graphics*, 25(4):175–186, 1991.

[ICG86] D. S. Immel, M. F. Cohen, and D. P. Greenberg. A radiosity method for non-diffuse environments. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, pages 133–142, 1986.

[JC95] H. W. Jensen and N. J. Christensen. Photon maps in bidirectional Monte Carlo ray tracing of complex objects. *Computers and Graphics*, 19(2):215–224, 1995.

[JC98] H. W. Jensen and P. H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. *Computers and Graphics (SIGGRAPH '98 Proceedings)*, pages 311–320, 1998.

[Jen95] H. W. Jensen. Importance driven path tracing using the photon maps. In *Rendering Techniques '95*, pages 326–335, 1995.

[Jen96] H. W. Jensen. Global illumination using photon maps. In *Rendering Techniques '96*, pages 21–30, 1996.

[JGMHe88] K. I. Joy, C. W. Grant, N. L. Max, and Lansing Hatfield (editors). *Computer Graphics: Image Synthesis.* IEEE Computer Society Press, Los Alamitos, CA., 1988.

[Kaj86] J. T. Kajiya. The rendering equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, pages 143–150, 1986.

[Kel95] A. Keller. A quasi-Monte Carlo algorithm for the global illumination in the radiosity setting. In H. Niederreiter and P. Shiue, editors, *Monte-Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pages 239–251. Springer, 1995.

[Kel96a] A. Keller. Quasi-Monte Carlo Radiosity. In X. Pueyo and P. Schröder, editors, *Rendering Techniques '96 (Proc. 7th Eurographics Workshop on Rendering)*, pages 101–110. Springer, 1996.

[Kel96b] A. Keller. The fast Calculation of Form Factors using Low Discrepancy Sequences. In *Proc. Spring Conference on Computer Graphics (SCCG '96)*, pages 195–204, Bratislava, Slovakia, 1996. Comenius University Press.

[Kel97] A. Keller. Instant radiosity. *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 49–55, 1997.

[Knu81] D.E. Knuth. *The art of computer programming. Volume 2 (Seminumerical algorithms).* Addison-Wesley, Reading, Mass. USA, 1981.

[Kón85] A. Kónya. *Fizikai kézikönyv műszakiaknak.* Műszaki Könyvkiadó, Budapest, 1985.

[Kra89] G. Krammer. Notes on the mathematics of the phigs output pipeline. *Computer Graphics Forum*, 8(8):219–226, 1989.

[Lan91] B. Lantos. *Robotok Irányitása*. Akadémiai Kiadó, Budapest, Hungary, 1991. (in Hungarian).

[LB94] B. Lange and B. Beyer. Rayvolution: An evolutionary ray tracing algorithm. In *Photorealistic Rendering Techniques*, pages 136–144, 1994.

[Lep80] G. P. Lepage. An adaptive multidimensional integration program. Technical Report CLNS-80/447, Cornell University, 1980.

[Lew93] R. Lewis. Making shaders more physically plausible. In *Rendering Techniques '93*, pages 47–62, 1993.

[LW93] E. Lafortune and Y. D. Willems. Bi-directional path-tracing. In *Compugraphics '93*, pages 145–153, Alvor, 1993.

[LW94] E. Lafortune and Y. D. Willems. Using the modified phong reflectance model for physically based rendering. Technical Report RP-CW-197, Department of Computing Science, K.U. Leuven, 1994.

[LW96] E. Lafortune and Y. D. Willems. A 5D tree to reduce the variance of Monte Carlo ray tracing. In *Rendering Techniques '96*, pages 11–19, 1996.

[Mát81] L. Máté. *Funkcionálanalízis műszakiaknak.* Műszaki Könyvkiadó, Budapest, 1981.

[Min41] M. Minnaert. The reciprocity principle in lunar photometry. *Astrophysical Journal*, 93:403–410, 1941.

[Mit92] D. Mitchell. Ray Tracing and Irregularities of Distribution. In *Rendering Techniques '92 (Proc. 3rd Eurographics Workshop on Rendering)*, pages 61–69, Bristol, UK, 1992.

[Mit96] D. P. Mitchell. Consequences of stratified sampling in graphics. *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 277–280, 1996.

[MRR$^+$53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[Nem90] A. Nemcsics. *Színdinamika, színes környezet mérése*. BME, Budapest, 1990.

[Neu95] L. Neumann. Monte Carlo radiosity. *Computing*, 55:23–42, 1995.

[NFKP94] L. Neumann, M. Feda, M. Kopp, and W. Purgathofer. A new stochastic radiosity method for highly complex scenes. In *Proc. of the 5th. EG Workshop on Rendering*, 1994.

[Nie92] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods.* SIAM, Pennsilvania, 1992.

[NNB97] L. Neumann, A. Neumann, and P. Bekaert. Radiosity with well distributed ray sets. *Computer Graphics Forum (Eurographics'97)*, 16(3):261–270, 1997.

[NNSK98a] L. Neumann, A. Neumann, and L. Szirmay-Kalos. Analysis and pumping up the albedo function. Technical Report TR-186-2-98-20, Institute of Computer Graphics, Vienna University of Technology, 1998. www.cg.tuwien.ac.at/.

[NNSK98b] L. Neumann, A. Neumann, and L. Szirmay-Kalos. New simple reflectance models for metals and other specular materials. Technical Report TR-186-2-98-17, Institute of Computer Graphics, Vienna University of Technology, 1998. www.cg.tuwien.ac.at/.

[NNSK99a] L. Neumann, A. Neumann, and L. Szirmay-Kalos. Compact metallic reflectance models. *Computer Graphics Forum (Eurographics'99)*, 18(3):161–172, 1999.

[NNSK99b] L. Neumann, A. Neumann, and L. Szirmay-Kalos. Reflectance models with fast importance sampling. *Computer Graphics Forum*, 18(4):249–265, 1999.

[NPT$^+$95] L. Neumann, W. Purgathofer, R. F. Tobler, A. Neumann, P. Elias, M. Feda, and X. Pueyo. The stochastic ray method for radiosity. In *Rendering Techniques '95*, pages 206–218, 1995.

[ON94] M. Oren and S. Nayar. Generalization of lambert's reflectance model. *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 239–246, 1994.

[Pel97] M. Pellegrini. Monte Carlo approximation of form factors with error bounded a priori. *Discrete and Computational Geometry*, 1997.

[PFTV92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C (Second Edition).* Cambridge University Press, Cambridge, USA, 1992.

[Pho75] B. T. Phong. Illumination for computer generated images. *Communications of the ACM*, 18:311–317, 1975.

[PM95] S. N. Pattanik and S. P. Mudur. Adjoint equations and random walks for illumination computation. *ACM Transactions on Graphics*, 14(1):77–102, 1995.

[PP98] J. Prikryl and W. Purgathofer. Perceptually based radiosity. In *Eurographics '98, STAR — State of the Art Report*, 1998.

[PTG95] W. Purgathofer, R. Tobler, and M. Geiler. Improved threshold matrices for ordered dithering. In Alan W. Paeth, editor, *Graphics Gems V*, pages 297–301. Academic Press, Boston, 1995.

[Rén62] Alfréd Rényi. *Wahrscheinlichkeitsrechnung.* VEB Deutscher Verlag der Wissenschaften, Berlin, 1962.

[Rén81] A. Rényi. *Valószinűségszámítás*. Tankönyvkiadó, Budapest, Hungary, 1981. (in Hungarian).

[RVW98] G. Renner, T. Várady, and V. Weiss. Reverse engineering of free-form features. In *PROLAMAT 98, CD proceedings*, Trento, 1998.

[SAWG91] F. X. Sillion, J. R. Arvo, S. H. Westin, and D. P. Greenberg. A global illumination solution for general reflectance distributions. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):187–198, 1991.

[Sbe96] M. Sbert. *The Use of Global Directions to Compute Radiosity.* PhD thesis, Catalan Technical University, Barcelona, 1996.

[Sbe99] M. Sbert. Optimal absorption probabilities for random walk radiosity. *to be published*, 1999.

[SC94] F. Sillion and Puech C. *Radiosity and Global Illumination.* Morgan Kaufmann Publishers, Inc., San Francisco, 1994.

[Sch93] Ch. Schlick. A customizable reflectance model for everyday rendering. In *Fourth Eurographics Workshop on Rendering*, pages 73–83, Paris, France, 1993.

[Sch96] J. Schanda. CIE colorimetry and colour displays. In *IS&T/SID Conf. Scottsdale*, 1996.

[SDS95] F. Sillion, G. Drettakis, and C. Soler. Clustering algorithm for radiance calculation in general environments. In *Rendering Techniques '95*, pages 197–205, 1995.

[Se66] J. Shrider (editor). *The Monte-Carlo Method.* Pergamon Press, Oxford, 1966. Also in *The method of statistical trials (The Monte Carlo Method)*, Fizmatgiz, Moscow, 1965.

[SGCH94] P. Schröder, S.J. Gortler, M.F. Cohen, and P. Hanrahan. Wavelet projections for radiosity. *Computer Graphics Forum*, 13(2):141–151, 1994.

[SH81] R. Siegel and J. R. Howell. *Thermal Radiation Heat Transfer.* Hemisphere Publishing Corp., Washington, D.C., 1981.

[Shi90] P. Shirley. A ray-tracing method for illumination calculation in diffuse-specular scenes. In *Proc. Graphics Interface*, pages 205–212, 1990.

[Shi91a] P. Shirley. Discrepancy as a quality measure for sampling distributions. In *Eurographics '91*, pages 183–194. Elsevier Science Publishers, 1991.

[Shi91b] P. Shirley. Time complexity of Monte-Carlo radiosity. In *Eurographics '91*, pages 459–466. Elsevier Science Publishers, 1991.

[SK95] L. Szirmay-Kalos. Stochastic sampling of two-dimensional images. In *COMPUGRAPHICS '95*, Alvor, 1995.

[SK98a] L. Szirmay-Kalos. Global ray-bundle tracing. Technical Report TR-186-2-98-18, Institute of Computer Graphics, Vienna University of Technology, 1998. www.cg.tuwien.ac.at/.

[SK98b] L. Szirmay-Kalos. Stochastic iteration for non-diffuse global illumination. Technical Report TR-186-2-98-21, Institute of Computer Graphics, Vienna University of Technology, 1998. www.cg.tuwien.ac.at/.

[SK99a] L. Szirmay-Kalos. *Monte-Carlo Methods in Global Illumination.* Institute of Computer Graphics, Vienna University of Technology, Vienna, 1999.

[SK99b] L. Szirmay-Kalos. Stochastic iteration for non-diffuse global illumination. *Computer Graphics Forum (Eurographics'99)*, 18(3):233–244, 1999.

[SK99c] L. Szirmay-Kalos. *Számítógépes grafika.* ComputerBooks, Budapest, 1999.

[SKCP98] L. Szirmay-Kalos, B. Csébfalvi, and W. Purgathofer. Importance-driven quasi-Monte Carlo solution of the rendering equation. In *Winter School of Computer Graphics '98*, pages 377–386, Plzen, Czech Republic, 1998.

[SKCP99] L. Szirmay-Kalos, B. Csébfalvi, and W. Purgathofer. Importance driven quasi-random walk solution of the rendering equation. *Computers and Graphics*, 23(2):203–212, 1999.

[SKDP99] L. Szirmay-Kalos, P. Dornbach, and W. Purgathofer. On the start-up bias problem of metropolis sampling. In *Winter School of Computer Graphics '99*, pages 273–280, Plzen, Czech Republic, 1999.

[SKe95] L. Szirmay-Kalos (editor). *Theory of Three Dimensional Computer Graphics.* Akadémia Kiadó, Budapest, 1995. http://www.iit.bme.hu/˜szirmay.

[SKF97] L. Szirmay-Kalos and T. Fóris. Radiosity algorithms running in sub-quadratic time. In *Winter School of Computer Graphics '97*, pages 562–571, Plzen, Czech Republic, 1997.

[SKFNC97] L. Szirmay-Kalos, T. Fóris, L. Neumann, and B. Csébfalvi. An analysis to quasi-Monte Carlo integration applied to the transillumination radiosity method. *Computer Graphics Forum (Eurographics'97)*, 16(3):271–281, 1997.

[SKFP98a] L. Szirmay-Kalos, T. Fóris, and W. Purgathofer. Non-diffuse, random-walk radiosity algorithm with linear basis functions. *Machine Graphics and Vision*, 7(1):475–484, 1998.

[SKFP98b] L. Szirmay-Kalos, T. Fóris, and W. Purgathofer. Quasi-Monte Carlo global ray-bundle tracing with infinite number of rays. In *Winter School of Computer Graphics '98*, pages 386–393, Plzen, Czech Republic, 1998.

[SKM95] L. Szirmay-Kalos and G. Márton. On convergence and complexity of radiosity algorithms. In *Winter School of Computer Graphics '95*, pages 313–322, Plzen, Czech Republic, 14–18 February 1995. http//www.iit.bme.hu/˜szirmay.

[SKP98] L. Szirmay-Kalos and W. Purgathofer. Global ray-bundle tracing with hardware acceleration. In *Rendering Techniques '98*, pages 247–258, 1998.

[SKP99] L. Szirmay-Kalos and W. Purgathofer. Global ray-bundle tracing with infinite number of rays. *Computers and Graphics*, 23(2):193–202, 1999.

[SMP98] M. Sbert, R. Martinez, and X. Pueyo. Gathering multi-path: a new Monte-Carlo algorithm for radiosity. In *Winter School of Computer Graphics '98*, pages 331–338, Plzen, Czech Republic, 1998.

[Sob91] I. Sobol. *Die Monte-Carlo Methode.* Deutscher Verlag der Wissenschaften, 1991.

[SP89] F. Sillion and C. Puech. A general two-pass method integrating specular and diffuse reflection. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, pages 335–344, 1989.

[SPNP96] M. Sbert, X. Pueyo, L. Neumann, and W. Purgathofer. Global multipath Monte Carlo algorithms for radiosity. *Visual Computer*, pages 47–61, 1996.

[SPS98] M. Stamminger, Slussalek P., and H-P. Seidel. Three point clustering for radiance computations. In *Rendering Techniques '98*, pages 211–222, 1998.

[ST93] G. Stoyan and G. Takó. *Numerikus mószerek.* ELTE, Budapest, 1993.

[SW93] P. Shirley and C. Wang. Luminaire sampling in distribution ray tracing. *SIGGRAPH '93 Course Notes*, 1993.

[SWZ96] P. Shirley, C. Wang, and K. Zimmerman. Monte Carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, 1996.

[Tam92] F. Tampieri. Accurate form-factor computation. In David Kirk, editor, *Graphics Gems III*, pages 329–333. Academic Press, Boston, 1992.

[Vea97] E. Veach. *Robust Monte Carlo Methods for Light Transport Simulation.* PhD thesis, Stanford University, http://graphics.stanford.edu/papers/veach_thesis, 1997.

[VG95] E. Veach and L. Guibas. Bidirectional estimators for light transport. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 419–428, 1995.

[VG97] E. Veach and L. Guibas. Metropolis light transport. *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 65–76, 1997.

[VMC97] T. Várady, R. R. Martin, and J. Cox. Reverse engineering of geometric models - an introduction. *Computer-Aided Design*, 29(4):255–269, 1997.

[War92] G. Ward. Measuring and modeling anisotropic reflection. *Computer Graphics*, 26(2):265–272, 1992.

[War95] T. Warnock. Computational investigations of low-discrepancy point sets. In H. Niederreiter and P. Shiue, editors, *Monte-Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, pages 354–361. Springer, 1995.

[WCG87] J. R. Wallace, M. F. Cohen, and D. P. Greenberg. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 311–324, 1987.

[WS82] G. Wyszecki and W. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae.* Wiley, New York, 1982.

[ZS95] K. Zimmerman and P. Shirley. A two-pass solution to the rendering equation with a source visibility preprocess. In *Rendering Techniques '95*, pages 284–295, 1995.

SUBJECT INDEX