MONTE CARLO PATH TRACING FOR PHOTOREALISM

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF SCIENCE

By

David J Hutchinson

Department of Computer Science

October 1993

Contents

List of Figures

List of tables

Abstract

UNIVERSITY OF MANCHESTER

ABSTRACT OF THESIS submitted by **David J Hutchinson** for the Degree of Master of Science and entitled **Monte Carlo Path Tracing for Photorealism**

Month and Year of Submission: October 1993

The production of photorealistic pictures in computer graphics by modelling light interaction has always been the topic of much research. In 1986, Kajiya introduced the rendering equation which accounted for all of the light transport within a scene. This equation has been used by the author to implement a path tracer, which is capable of producing realistic images of complex environments. The images produced exhibit umbra and penumbra regions, diffuse inter-object reflections, hazy reflections, and caustics. Multiple light sources, including area sources, complex shapes, and texture mapping have also been included. Speed up techniques include bounding boxes, composite objects, and control over the number of rays shot per pixel, as well as the number of higher generation rays.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Chapter 1

Introduction

In the production of photorealistic images, one of the more popular techniques used is that of light-backward ray tracing [51]. This involves casting rays from a viewing point, through a virtual screen, and intersecting them with a scene stored in a data structure. The intensity of the light transmitted from the point of intersection along the direction towards the viewing point is then calculated and converted into a form suitable for display. This is demonstrated in figure 1.1.

Conventional ray tracing suffers however from its inability to model correctly the light scattered due to diffuse inter-object reflections [49]. This is because in calculating the intensity of the light transmitted from an intersection point, only contributions from the specular and transmitted directions are considered, together with another from the lighting present, shown in figure 1.2. The contribution from diffuse inter-object reflections within a scene is approximated by a constant lighting term, resulting in pictures which look far from realistic.

The rendering equation, introduced by Kajiya [29], correctly models the diffuse

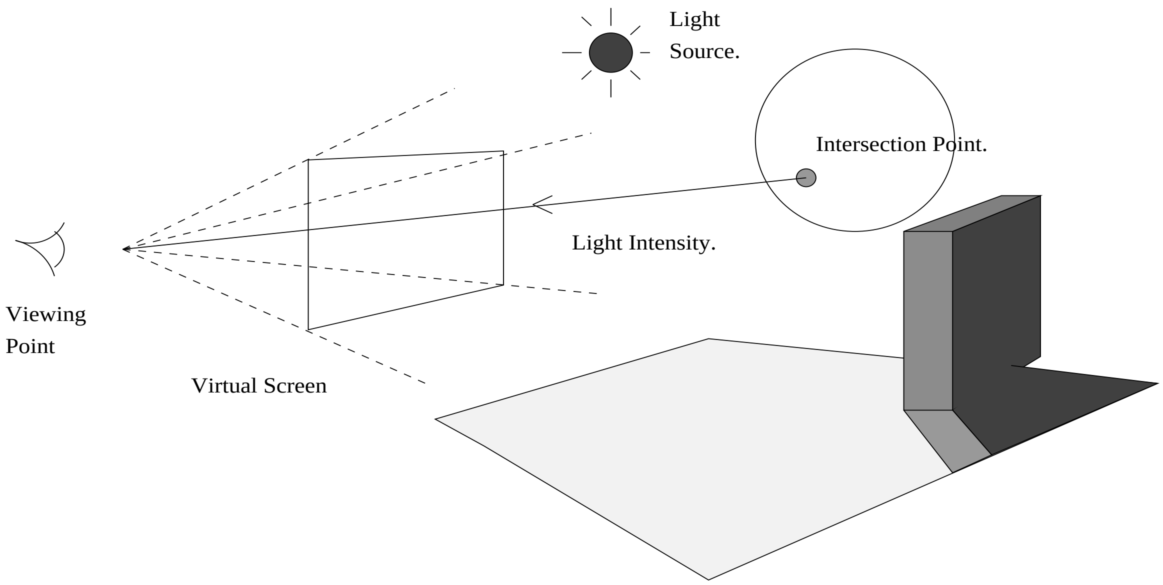


Figure 1.1: Ray Intersecting with a Scene

inter-object reflections within a scene by taking an integral of the light intensity over all surfaces. He then continued to suggest a view dependent solution to this equation which involved casting rays through the scene from the viewing point as for normal ray tracing, but at each intersection point only one ray is followed in either the diffuse, specular, or transmitted direction. This results in what is known as a *path* through the scene. By choosing which sort of ray to follow at each ray intersection point, path tracing spends less time following unimportant ray directions, unlike ray tracing which spends most of its time tracing high generation rays which do not contribute greatly to the final intensity. The problem with path tracing is that in order to model diffuse inter-object reflections correctly, many rays must be followed per pixel in order to reduce the variance across an image, that is to eliminate speckling effects. Kajiya [29] reported using a standard forty rays per pixel to produce his sample image, but gave no indication as to how much extra sampling was used to model the diffuse contribution, or how much might be needed for a badly illuminated scene. For

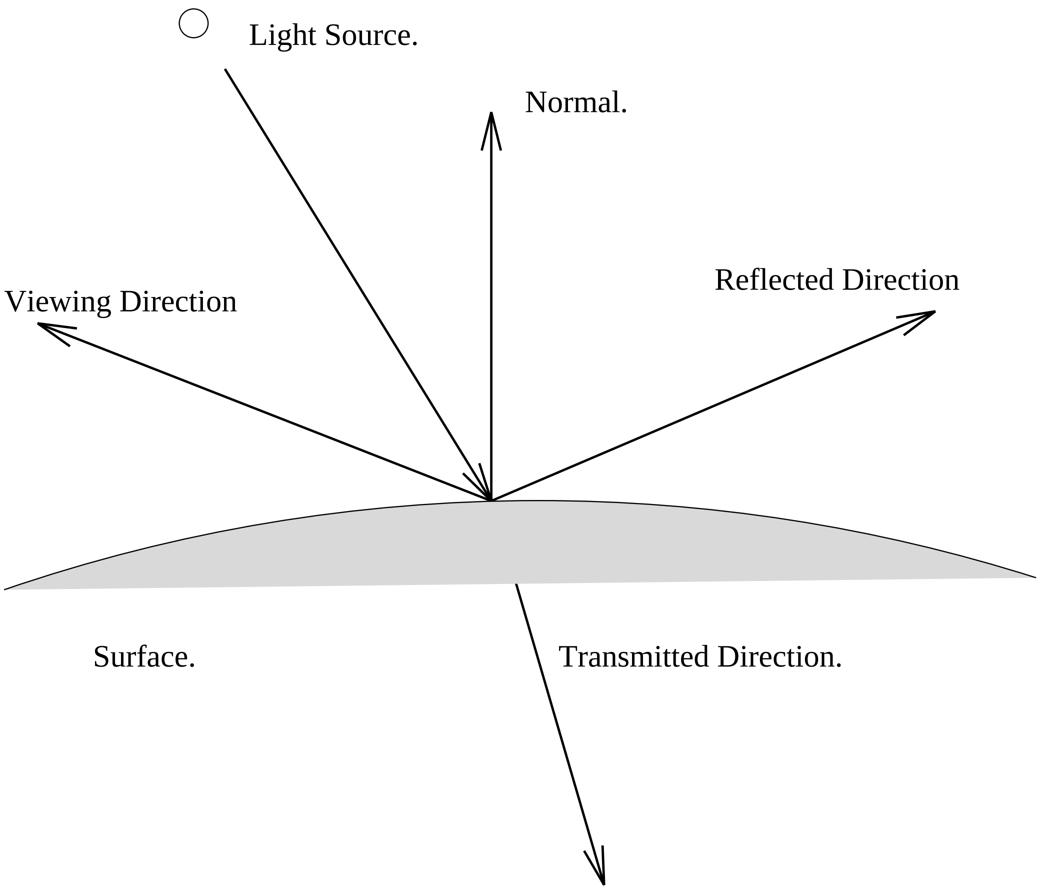


Figure 1.2: Light interaction with a surface

realistic scenes which may include a large amount of indirect lighting, variance reduction techniques must be included to control the number of rays cast per pixel, and also the number of higher generation rays followed per intersection. By controlling these variables, more work can be done in interesting portions of an image, and less work where colour is uniform.

This thesis investigates path tracing techniques as applied to directly lit scenes, and also to those which consist of a large amount of indirect lighting. Images containing complex and composite objects have also been examined. Chapter two outlines the illumination models which have been used in ray tracing algorithms, and then continues further to discuss how these models can be applied within a global illumination model. In chapter three, the sampling techniques used to reduce variance across an image and decrease rendering time are discussed, with particular attention paid to those methods which are used inside this implementation of the path tracer, coded entirely by the author. Chapter four looks at ways in which the path tracer can be extended to produce realistic images, and chapter five presents the results and conclusions.

Chapter 2

Illumination and Illumination Models

Before the specifics of path tracing can be discussed, an outline of some of the models available for illuminating a scene will be presented.

2.1 Light Transport

The way in which light interacts with a surface is very complex, and so many techniques have been developed over the years to model it [6], [7], [11], [23], [39], [47]. When a photon is incident upon a surface atom, one of a number of things may happen to it, the probability of which depends on the surface properties. The photon can be completely absorbed by the surface and transformed into heat, or absorbed and then re-radiated in a random direction. Some photons do not interact with the surface itself, but have their direction perturbed, and can

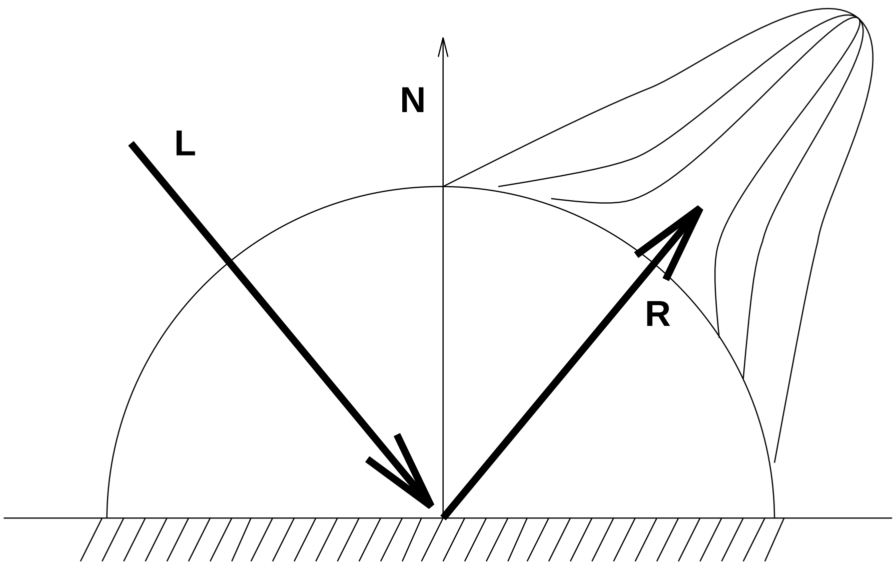


Figure 2.1: Light Distribution for a Surface

either be reflected off or transmitted through the surface. A discussion of the complete absorption of a photon is not included.

The way in which photons interact with materials may be described using probability functions, however a distribution of the most probable directions in which the light will be emitted, or transmitted, can be found for most materials. The distribution for a typical surface without transmissive properties included, is shown in figure 2.1. It can be seen that there is a spike in the distribution in the reflected direction **R**. At this point, the probability that a photon will be reflected is highest. The hemispherical distribution represents that light which has been absorbed and re-emitted. Most materials, except those with very shiny or peculiar surface properties exhibit this to some extent. Similar properties are found for transmitting surfaces, with a peak occurring in the direction in which photons are scattered through the surface without being absorbed.

In order to calculate the amount of light emitted in a certain direction, we could turn to physics to provide the exact physical equations for a given surface and an incident photon. The equations however, are not in a form soluble by computer techniques, and also the interaction of individual photons with a surface cannot be easily modelled, making an exact physical model of this type unsatisfactory.

In order to make light transport suitable for computation, many approximate models have been developed. The way in which most of these work is not to look at individual photon transport, but to consider the time averaged case. That is to assume that when we are looking at a scene, all photon interaction has come to an end, and the picture we are looking at is in equilibrium, and will not change over time.

2.2 Approximating the Light Distribution

A common way of approximating the light leaving a surface is to split the distribution up into three quantities. They are the *diffuse* $I_{dr}(\lambda)$, *specular* $I_{sr}(\lambda)$, and *transmitted* $I_{st}(\lambda)$ components, making the final intensity $I(\lambda)$ leaving a surface equal to the sum of these three.

2.2.1 Diffuse Reflection

Diffuse reflection takes into account the absorption and re-radiation of light at a surface. Most materials exhibit this aspect of light emission to some extent, except for the most shiny of surfaces. If a photon is re-radiated after being absorbed, there is nothing which determines in which direction the photon will be emitted. Any one photon will be emitted in one random direction, however

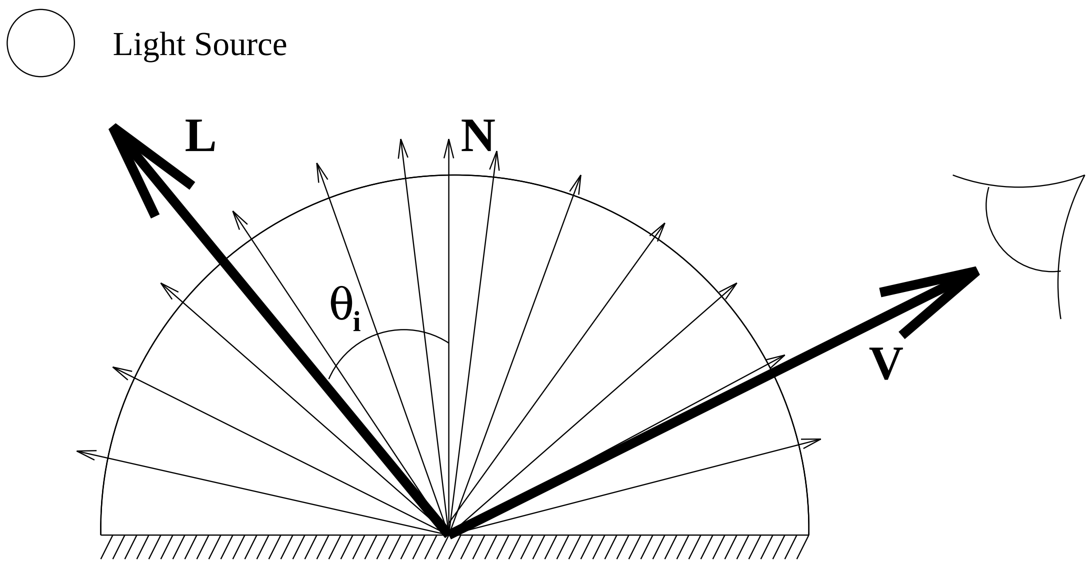


Figure 2.2: Diffuse Reflection

over a period of time, photons will be emitted randomly in all directions, and if averaged over a long period of time will be emitted equally in all directions, with equal intensity. The only geometry that must be taken into account then, is how much of the surface is visible to the light source. The amount of light emitted by a surface is proportional to $\cos(\theta_i)$, the angle between the surface normal **N**, and the incident light ray **L**, as shown in figure 2.2. The wavelength intensities re-radiated compared to those absorbed due to the cos term are only significantly affected for angles of incidence greater than 60^o.

Since the same intensity of light is radiated in all directions, it does not matter from which direction we are viewing, the intensity $I_{dr}(\lambda)$ of the light diffusely reflected at wavelength λ, in the viewing direction **V** is given by:

$$I_{dr}(\lambda) = \kappa_{dr}(\lambda) I_i(\lambda)(\hat{\mathbf{N}} \cdot \hat{\mathbf{L}})$$

where $\kappa_{dr}(\lambda)$ is the diffuse reflection coefficient at this wavelength, and $I_i(\lambda)$ is the intensity of the incident ray. The diffuse reflection coefficient is the extent to which the surface reflects diffuse light, and may be different over the whole

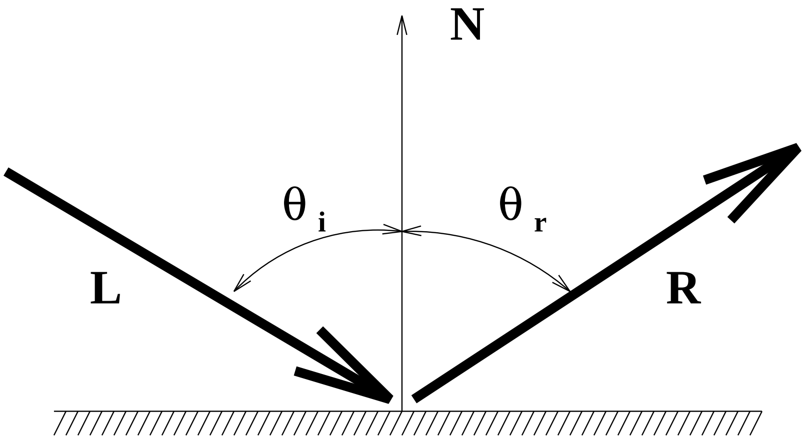


Figure 2.3: Specular Reflection

spectrum of wavelengths. For example, a mirror might have a diffuse reflectance of close to 0, while a diffuse surface such as matt paint would be close to 1. Surfaces which are perfectly diffuse are known as Lambertian surfaces.

2.2.2 Specular Reflection

If a light ray is incident upon a surface and *specular reflection* occurs, the light bounces off the top of the surface, and is not subject to absorption or re-radiation. This is shown in figure 2.3, where the angle θ_i between the surface normal **N**, and the incident ray **L**, is equal to the angle θ_r, between **N** and the reflected ray **R**. This sort of reflection can be seen when looking into a mirror, and also in the form of highlights on a shiny surface.

Several methods are available for calculating the intensity of light reflected off a surface in the viewing direction. A simple way was first developed by Phong [39], who noted that light incident upon a surface with normal **N**, along the direction **L**, will reflect the ray about the reflection vector **R**. This depends on the degree to which an object specularly reflects, and results in differently sized highlights appearing on a surface. The more shiny an object is, the smaller a highlight will be. If an observer is looking down the reflection vector $\mathbf{R}$ at a shiny surface, then the observed intensity would be very large, however as the eye point moves away from $\mathbf{R}$ by an angle θ, the intensity of the highlight will drop off quickly. This drop off was modelled by Phong using a $\cos^n \theta$ term, where n is the materials *specular reflection highlight coefficient.* This coefficient is small for non specular materials, and tends to infinity for a perfect reflector. This term is included into the equation for the specularly reflected intensity $I_{sr}(\lambda)$ at wavelength λ in the following way:

$$I_{sr}(\lambda) = \kappa_{sr}(\lambda) I_i(\lambda)(\hat{\mathbf{N}} \cdot \hat{\mathbf{H}})^n$$

where $I_i(\lambda)$ is the intensity of the incident light, $\hat{\mathbf{H}}$ is the perfect specular reflection microfacet normal for the surface, and $\kappa_{sr}(\lambda)$ is the specular reflection coefficient. This is illustrated in figure 2.4. $\hat{\mathbf{H}}$ can be calculated using the formula:

$$\hat{\mathbf{H}} = \frac{\hat{\mathbf{L}} + \hat{\mathbf{V}}}{|\hat{\mathbf{L}} + \hat{\mathbf{V}}|}$$

where $\hat{\mathbf{V}}$ is the viewing vector.

Other notable reflection models which use more physically based methods to calculate the reflected intensity [47], [11], attempt to model surface roughness as a set of randomly orientated microfacets, figure 2.5. A distribution function for a surface is calculated, which is an indication of the number of microfacets which are parallel to the perfect specular microfacet, and so is a measure of how rough a surface will be. The intensities calculated by these methods appear physically correct, and for surfaces with a large Phong exponent, Blinn [6] concluded that they are no more computationally expensive. It is the case though, that for large values of the Phong exponent, where the number of multiplications becomes

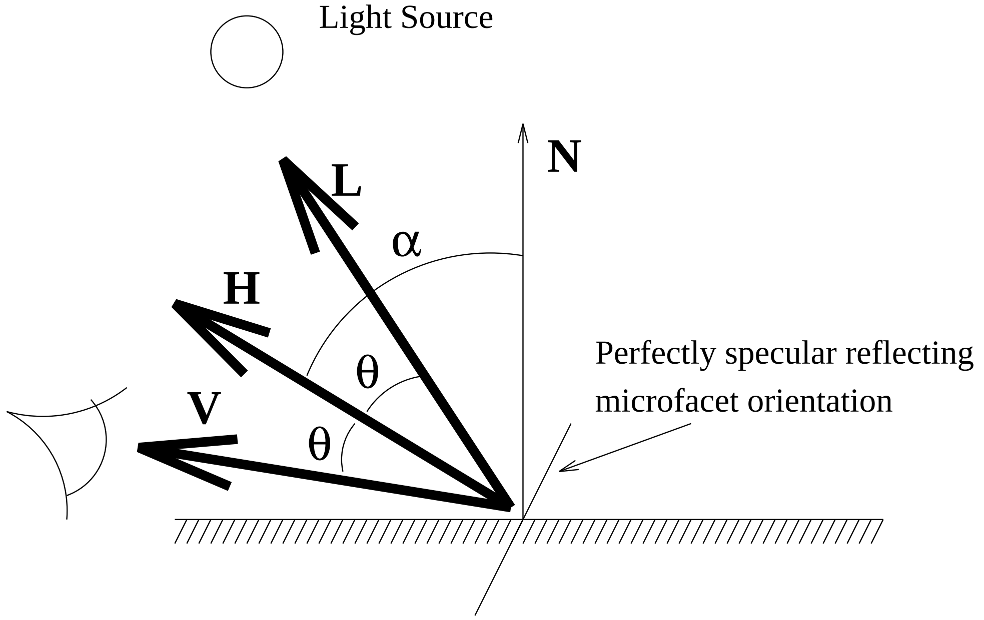


Figure 2.4: Specular Reflection

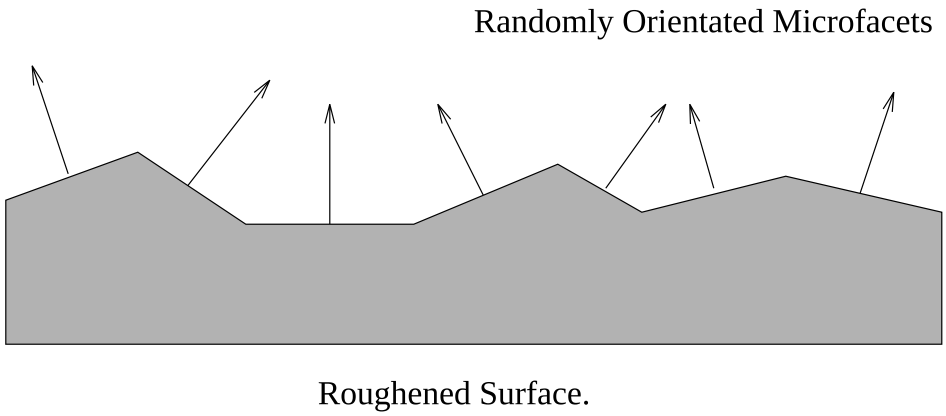


Figure 2.5: Randomly Distributed Microfacets

prohibitive, an approximation may be made, and for this reason the Phong model has been used.

2.2.3 Specular Transmission

If a surface exhibits *specular transmission*, then light may be incident from behind it and pass through, contributing to the light leaving the surface. The media on both sides of a boundary (or *interface*) can be the same, such as passing through

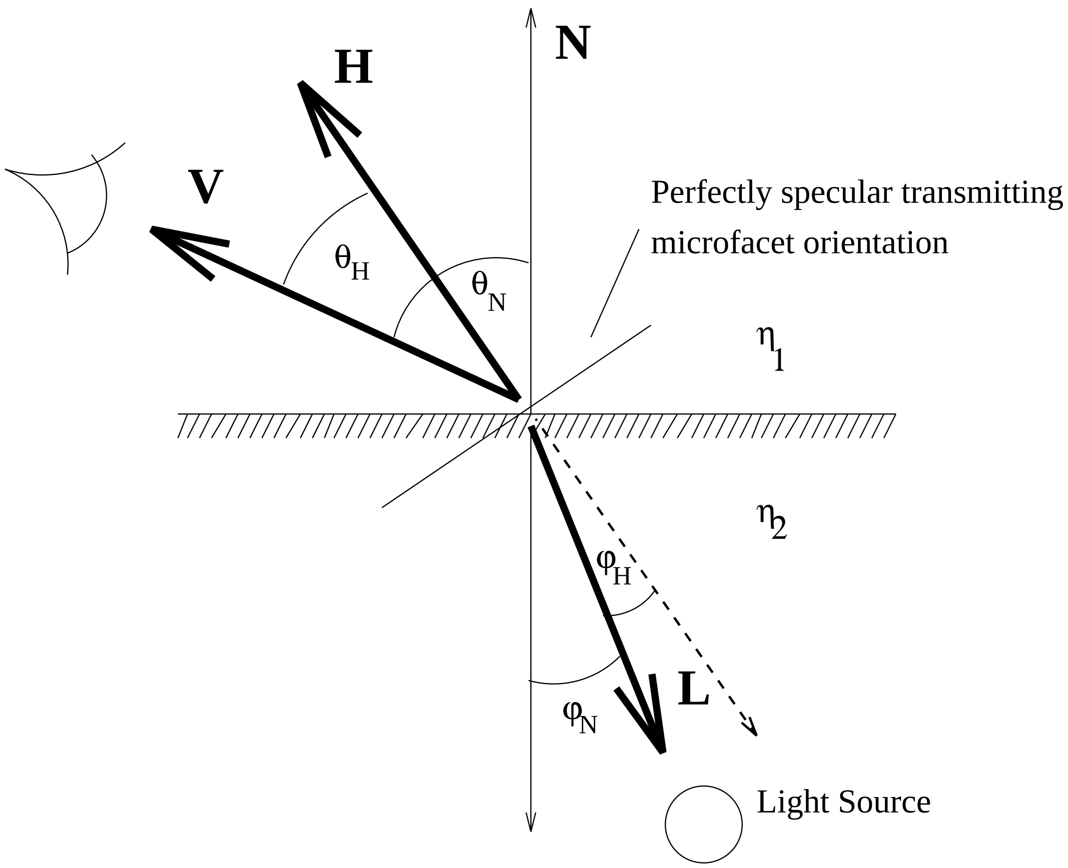


Figure 2.6: Specular Transmission

air, or different, such as passing from air to water. An interaction of this type can cause the apparent direction of the ray to change, due to the bending (or *refraction*) of the light rays as they pass between media.

Each medium has an *index of refraction*, which is the ratio of the speed of light in that media, to the speed of light in a vacuum, and is always greater than or equal to unity. The amount to which the light is refracted on crossing media is found by comparing the indices of refraction of the two media, and the angle of the incident light. The intensity of light transmitted through a surface $I_{tr}(\lambda)$ at wavelength λ is given by:

$$I_{st}(\lambda) = \kappa_{st}(\lambda) I_i(\lambda)(\hat{\mathbf{N}} \cdot \hat{\mathbf{H}})^n$$

where $I_i(\lambda)$ is the intensity of the incident light, $\hat{\mathbf{H}}$ is the perfect specular transmitting microfacet normal for the surface, and $\kappa_{st}(\lambda)$ is the specular transmissive coefficient.

Figure 2.8 shows an incident light ray marked $\mathbf{I}$ striking a surface and making an angle $\theta_{\mathbf{i}}$ with the normal $\mathbf{N}$. The transmitted light $\mathbf{T}$ makes an angle $\theta_{\mathbf{t}}$ (the *angle of refraction*), which can be determined using Snell's Law of refraction [30], which states that:

$$\frac{\sin\,\theta_i}{\sin\,\theta_t} = \eta_{21} = \frac{\eta_2}{\eta_1} \tag{2.1}$$

where:

- η_1 is the index of refraction of medium 1 with respect to vacuum.
- η_2 is the index of refraction of medium 2 with respect to vacuum.
- η_{21} is the index of refraction of medium 2 with respect to medium 1.

The calculation of the $\hat{\mathbf{H}}$ vector is more involved than for the reflected case. The following equations have been used to calculate it, and should be referenced against figure 2.6.

$$\theta_N = \arccos(\hat{\mathbf{V}} \cdot \hat{\mathbf{N}})$$

$$\phi_N = \arccos(\hat{\mathbf{L}} \cdot -\hat{\mathbf{N}})$$

The angles are co-planar so $\phi_{\mathbf{N}} + \phi_{\mathbf{H}} = \theta_{\mathbf{N}} - \theta_{\mathbf{H}}$, then if we let $A = \theta_{\mathbf{N}} - \phi_{\mathbf{H}}$ then $\phi_{\mathbf{H}} = A - \theta_{\mathbf{H}}$. Taking sin of both sides gives:

$$\sin(\phi_{\mathbf{H}}) = \sin(A)\cos(\theta_{\mathbf{H}}) - \cos(\theta_{\mathbf{H}})\sin(A)$$

Now by substituting Snell's Law into this, you can write:

$$(\alpha \cos(\theta_H) - \beta \sin(\theta_H)) = \sin(\theta_H)$$

where $\alpha = \eta_{21} \sin(A)$, and $\beta = \eta_{21} \cos(A)$. Now by simple manipulation the angle θ_H is given by

$$\theta_H = \arctan\left(\frac{\alpha}{1+\beta}\right)$$

from which **H** can be calculated.

The angle of refraction is dependent on the wavelength of the incident light, and gives rise to such phenomena as the splitting of white light into its individual components by a prism. This can present complications in tracing algorithms which will be mentioned later.

2.2.4 Attenuation of Light

When light travels through any medium its intensity is subject to attenuation by that medium. This results in similar objects appearing darker the further away they are from the viewing point, and so is an important aspect of modelling. For a point light source, intensity I_i, emitting light equally in all directions, at a distance d from a surface, the intensity I striking that surface is given by the inverse square law, discussed by Foley [15]:

$$I = \frac{I_i}{4\pi d^2}$$

However, Foley *et al* state that whilst this law is physically correct, it is rarely used in practice, as it results in very large intensity values close to the light source, and very small ones further away.

Foley *et al* go on to give the commonly used compromise function:

$$F_{att} = max\left(\frac{1}{c_1 + c_2 d + c_3 d^2}\right)$$

where c_1, c_2, c_3 are implementation dependent constants. This equation is not based on any physical observation, but is a convenient way of modelling the attenuation to give a more realistic looking image.

2.2.5 Illumination Models

There are two general ways of modelling the light transport within a scene. They are local and global illumination.

Local illumination models attempt to account for interactions between objects and light sources, but make no attempt to accurately model inter-object light transport. After light from a source has interacted with a surface, its path is no longer of any concern. The images produced by this method are not very realistic because of this, however they are sufficient for many applications, such as in engineering.

Global illumination models attempt to account for the interaction between all of the objects in a scene, and all of the light in a scene, whether that light has come directly from a light source, or has been reflected off or transmitted through an object. These models, if implemented sufficiently well will produce photorealistic images, and are the subject of the rest of this chapter.

2.3 Ray Tracing

Ray tracing [51], is a technique whereby light rays are followed from a viewing point, through a virtual viewing screen, until they either intersect with an object in a scene or not. Then, the intensity of light incident upon that point and emitted in the viewing direction is calculated, to find the final colour of a pixel, or whatever device is being used to view the scene. The idea of following rays from the viewing point through a scene is often known as *light-backward ray tracing*, whereas *light-forward ray tracing* [24], [2], [49] involves following rays from the light source, until they either hit or miss the viewing plane. Light-forward ray tracing takes a significant amount of time to compute, as most of the rays followed do not pass through the image plane. Figure 2.7 shows an example ray passing through a scene, as it is both reflected and transmitted at each intersection. At each intersection point the intensity of the light transmitted in the direction of the incident ray must be calculated. The total contribution of light coming from the reflected and transmitted directions are combined with that coming from the lighting present in the scene.

2.3.1 Whitted's Method

In order to model the interaction of light between objects in a scene, the direction in which light leaves a surface must be calculated. Whitted [51] derived the refraction formalae which accounted for the reflection and refraction of light between objects. Although some account of the diffuse inter-object reflections must be considered, the amount of time spent in calculating them within the ray

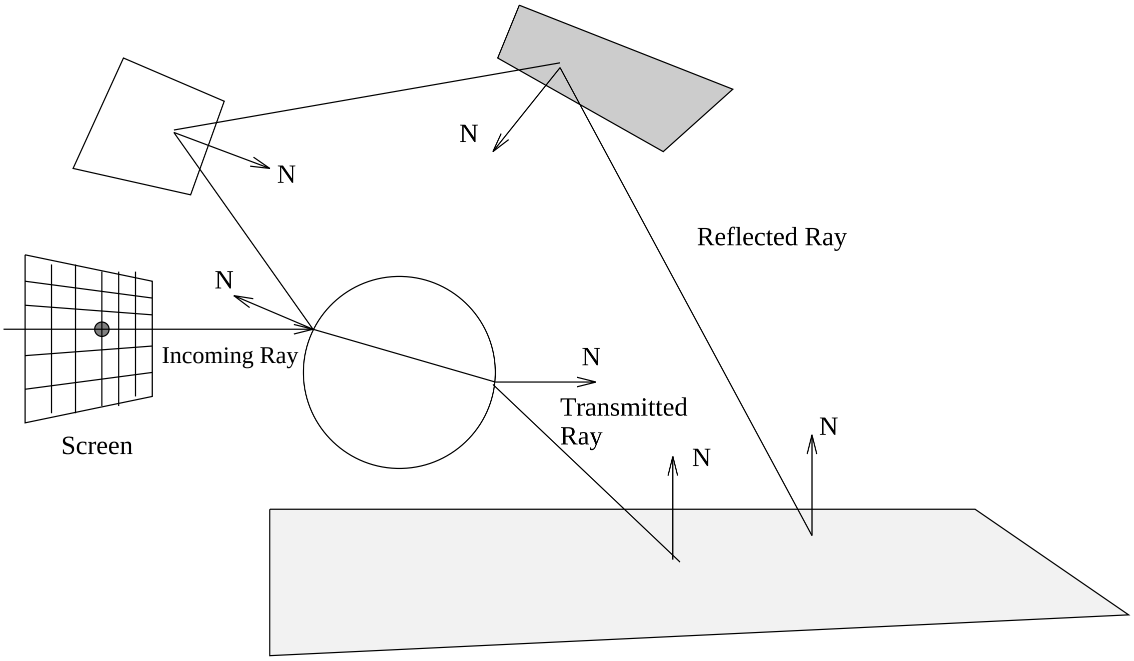


Figure 2.7: Ray Passing Through a Scene as for Ray Tracing

tracing model becomes too great. Whitted thus argued that the main types of light interaction between objects were those of reflection and refraction, and so at each intersection point only rays of this type are followed. A brief description of his model and the reflection and refraction formulae follows.

2.3.2 Ambient Light

Ambient light is one way of representing all of the diffusely reflected light in a scene. In a local illumination model, if a surface is not illuminated at all by any light source, then it will be rendered as black. However in most real scenes it would be illuminated in some way by diffuse reflections from other surfaces, and so in order to approximate this for the whole scene, an ambient term $I_{dr}(\lambda)$ is introduced to model the light intensity leaving a surface due to diffuse reflections from other objects, and is given by:

$$I_{dr}(\lambda) = I_a(\lambda)\kappa_{dr}$$

where $I_a(\lambda)$ is the ambient light intensity for the whole scene, and κ_{dr} is the diffuse reflection coefficient for the surface in question. This term takes no account of any directional dependencies, such as where an object is viewed from, or from which direction the ambient light is incident, and gives all points on a surface the same colour, making them appear flat. This is clearly unsatisfactory, and it will be shown later how this term may be omitted inside a path tracer.

2.3.3 Calculating the Reflected Direction

Given an incident ray **I** reflecting off a surface point with normal **N**, the direction of the reflected ray **R**, is found as follows.

There are two physical laws which can be used to find the reflected direction. The first is due to the fact that **R**, **I**, and **N** all lie in the same plane, and thus **R** must be some linear combination of **I** and **N**, so then we can write down the equation:

$$\hat{\mathbf{R}} = \alpha\hat{\mathbf{I}} + \beta\hat{\mathbf{N}}$$

where α and β are some constants. Also, the angle between the normal and the incident vector is the same as that between the normal and the reflected vector, so that:

$$\theta_i = \theta_r$$

Now by applying some simple transformations the reflected direction is given by:

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

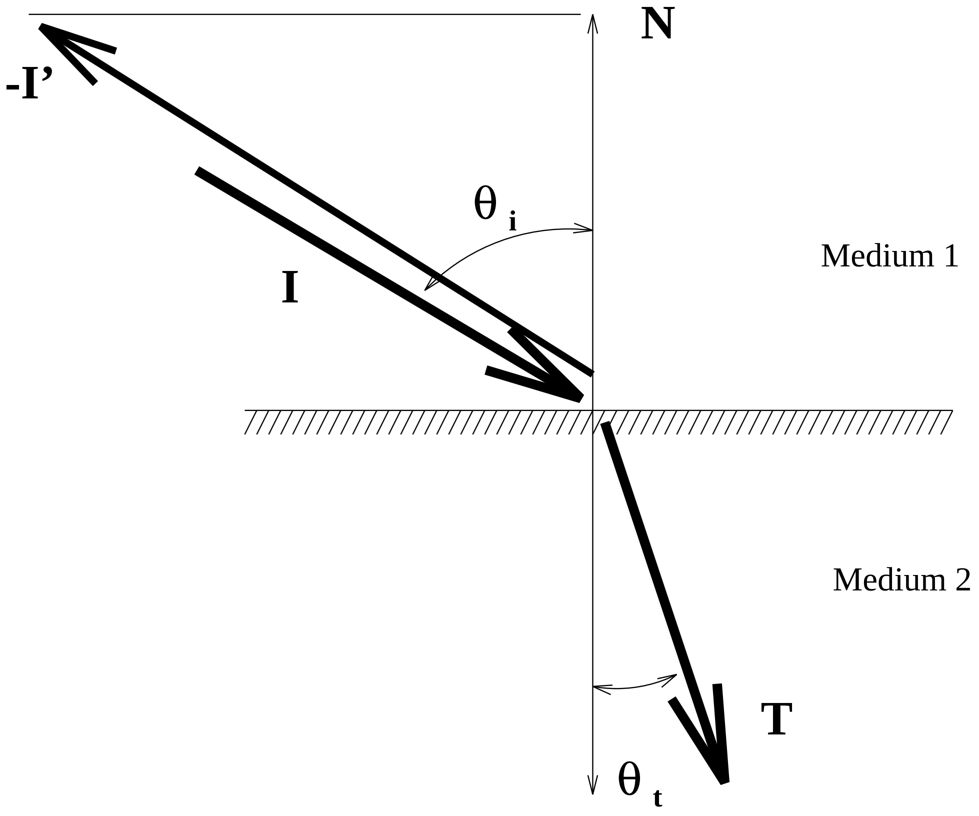


Figure 2.8: Specular Transmission

where all vectors are unit.

2.3.4 Calculating the Refracted Direction

As has already been said, the transmitted direction **T** is found by considering Snell's Law of refraction, which states that:

$$\frac{\sin\theta_i}{\sin\theta_t} = \eta_{21} = \frac{\eta_2}{\eta_1}$$

Whitted assumes that the normal $\hat{\mathbf{N}}$ is a unit vector, although the incident light vector **I** need not be. The first step is to find the vector $\mathbf{I}'$, which is a scaling of **I**, such that its projection onto $\hat{\mathbf{N}}$ is equal to $\hat{\mathbf{N}}$ (see figure 2.8). This yields:

$$\mathbf{I}' = \frac{\mathbf{I}}{-\mathbf{I}\cdot\hat{\mathbf{N}}}$$

which means that the vector $\mathbf{I}' + \hat{\mathbf{N}}$ is parallel to the surface. The refracted ray $\mathbf{T}$ can be written as a linear combination of this and the normal, such that $\mathbf{T} = \alpha(\mathbf{I}' + \hat{\mathbf{N}}) - \hat{\mathbf{N}}$ for some constant α, where $\mathbf{T}$ is not necessarily unit. The final step is to express α in terms of the vectors we know.

$$\alpha = (\eta_{21}^2 |\mathbf{I}'|^2 - |\mathbf{I}' + \hat{\mathbf{N}}|^2)^{-1/2}$$

If α is imaginary then total internal reflection occurs.

2.3.5 Final Remarks

Scenes rendered using Whitted's model produce very clean looking images, however due to the ambient term being included they still do not appear realistic. The reason for this is that in most real situations a scene will include a large amount of diffuse reflection, with different surfaces receiving a different amount of diffusely reflected light. In the search for photorealism, ray tracing does not provide an accurate enough approximation for the light transport, and other methods must be considered.

2.4 Radiosity

The many drawbacks of ray tracing led researchers to consider how the modelling of diffuse transport between surfaces might be better approximated. A new method based on the mathematics of heat transfer, which were already well understood, was developed by Goral *et al* [21], which accurately modelled the light transport between diffuse surfaces. A brief discussion of this technique follows.

The *radiosity* of a surface is a measure of the rate at which energy leaves that surface, and is used in computer graphics to calculate the distribution of light within a closed environment, or an open one which has been artificially enclosed by some bounding volume. The environment must be enclosed because radiosity relies on using the conservation of energy to model light transport. In applying radiosity theory it assumes that all surfaces are ideal opaque Lambertian emitters and/or reflectors, such that light is emitted or reflected equally in all directions. Radiosity has an advantage over conventional ray tracing techniques in that most real scenes consist of a large amount of diffusely reflected light which is not modelled by that method. Although radiosity works exclusively on this type of transport, effects caused by specular reflection and transmission may be added to the radiosity solution with little error. Another advantage is that the solution is view independent due to the nature of diffuse reflection, meaning that once the energy between surfaces in a scene has been calculated, any view may be found using a hidden surface removal algorithm.

The radiosity technique as described by Gatenby [16] is as follows. The surfaces inside a scene are split into a discrete number N of *patches*, and assuming that each patch is ideal Lambertian we can say:

$$A_i B_i = A_i E_i + \rho_i \sum_{j=1}^{N} B_j F_{ji} A_j$$

Which is the conservation of energy between all patches in the scene where:

- B_i is the radiosity of the i^{th} patch measured in energy per unit time per unit area ($Watts/m^2$), and is the energy leaving a patch of unit area in unit time.

- ρ_i is the reflectivity of the i^{th} patch, and is equal to the amount of energy reflected off a surface, relative to that which was incident upon it. It is a value between 0 and 1.

- F_{ij} is the *Form Factor* of the j^{th} patch relative to the i^{th} patch. It is equal to the fraction of energy that leaves patch j which arrives without interaction at patch i.

- E_i is the light emitted from the i_{th} patch.

- A_i is the surface area of the i_{th} patch, and is included since only unit area is mentioned in the equation.

All patches within the scene are treated as light sources, although some may have an emittance of zero. What is required is a solution to this equation for B_i, and this is found by calculating the form factors between every patch in the scene to give:

$$F_{ji} = \frac{1}{A_j}\int_{A_i}\int_{A_j}\frac{\cos\theta_i\cos\theta_j}{\pi r^2}H_{ji}dA_jdA_i$$

and similarly for F_{ij}. The *reciprocity relation for form factors* can be written as:

$$A_iF_{ij} = A_jF_{ji}$$

By applying the reciprocity equation to the equation above, every term has a common factor A_i which can be cancelled to leave.

$$B_i = E_i + \rho_i\sum_{j=1}^{N}B_jF_{ij}$$

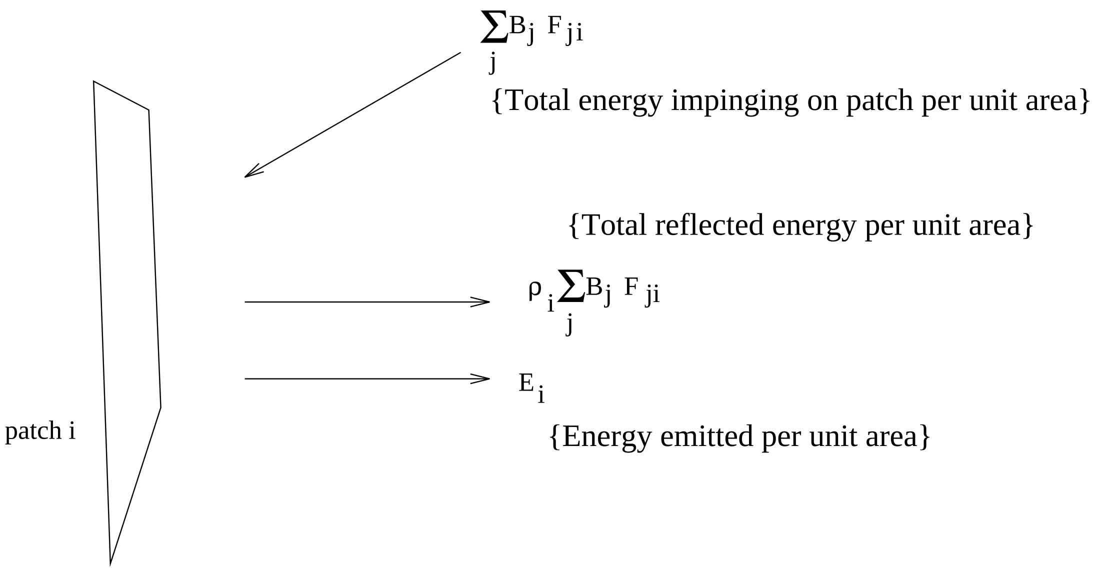


Figure 2.9: Energy Transfer at Patch i

This is shown in figure 2.9.

So then by applying this equation across all patches in a scene, the radiosity of each one may be found and stored. This information can be used to render the image from any viewing point [21], with the option of adding a ray tracing solution such that specular effects may be added [49].

2.5 Other Rendering Techniques

2.5.1 Beam Tracing

Ray tracing involves following infinitesimally small rays through the environment. A different solution which is discussed by Heckbert [25] called *beam tracing* sweeps areas through a scene to form *beams*. It works particularly well for polygonal models, since for this case the reflections are linear transformations, and refractions are often approximately so.

The beam tracer works by sweeping the viewing plane through the scene, and calculating beam surface intersections. A hidden surface algorithm is then applied to find the closest intersections along the beam. For each intersection, the beam is fragmented, and new beams created for the reflected and transmitted swaths of light. These new beams are redirected and recursively traced to create a beam tree which represents the entire picture.

2.5.2 Particle Tracing

Particle tracing [37] is based on the the model of light as a particle, as opposed to a wave description. Particles are emitted from light sources in a non-deterministic fashion, such that the number of particles emitted over time have a uniform flux across the source. Once emitted, the paths of the particles are followed until they interact with some object in the environment. The flux of particles incident upon a surface may be calculated, and the type of interaction which the particle is subject to (reflection, refraction or absorption) determined using Monte Carlo methods. As a result, the flux of particles leaving the surface, (i.e. the intensity of the light) via the different transport methods can be calculated. An advantage of this method is that participating volumes may be included.

2.6 Path Tracing

So far no technique has been described which accounts fully for all types of light interaction using standard wave descriptions. Kajiya [29] considered the way in which light transport was modelled, and introduced the rendering equation

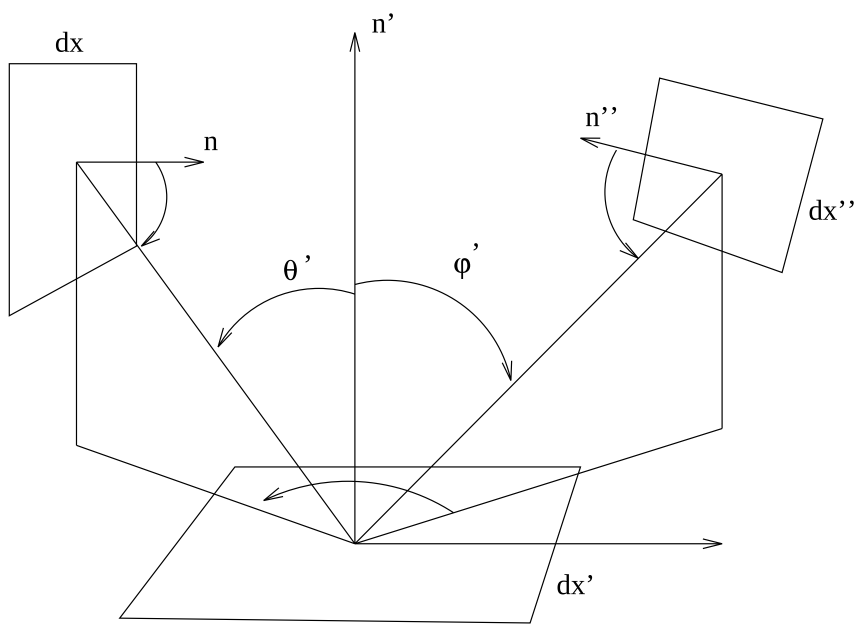


Figure 2.10: Equation Geometry

which correctly models all types of light transfer, by taking an integral of the light intensity over all surfaces. Kajiya then continued further to suggest a view dependent solution to this equation which would correctly model all types of light transport, and encompass the best aspects of both ray tracing and radiosity solutions.

2.6.1 The Rendering Equation

The rendering equation suggested by Kajiya [29] is of the form:

$$I(x,x') = g(x,x')\left[\epsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx''\right]$$

The equation states that the (*transport intensity*) of light from a surface point x' to point x, subject to attenuation by an amount g, is the sum of the emitted light ϵ from x' due to direct lighting, and the total light intensity $\int_S \rho I dx''$, which is scattered towards x from all other surface points x'' via x', .

Kajiya described these new terms in the equation as *unoccluded multipoint transport* quantities. A brief explanation of each of these quantities is now given. $I(x, x')$ (called the transport intensity) measures the energy of radiation passing from point x' to point x, and so the total energy reaching point x is given by $dE = I(x, x')dtdxdx'$ and is measured in units of $joule/m^4sec$.

The geometry term $g(x, x')$ is a measure of the occlusion of a surface point by other surface points. It is in fact a measure of the attenuation of the light between surfaces which has already been discussed. If a surface point is completely occluded then the geometry term will be equal to zero. This is similar to calculating the form factors in radiosity.

The term $\epsilon(x, x')$ is the *emittance*, and measures the energy emitted by a surface at point x', reaching point x, due to direct lighting. It is called the *unoccluded two point transport emittance* from x' to x, and the energy incident upon x is given by $dE = \frac{1}{r^2}\epsilon(x, x')dtdxdx'$, with units $joule/m^2sec$.

The scattering term $\rho(x, x', x'')$ is the amount of energy scattered by a surface element at x', originating from x'', and finally incident upon x. The energy incident upon x is given by $dE = \frac{1}{r^2}\rho(x, x', x'')I(x', x'')dtdxdx'dx''$. This term is a measure of the energy first incident upon x'' due to a light source, transmitted to the point x', which then transmits it to the point x. This term takes into account all forms of indirect illumination.

The rendering equation is a Fredholm equation of the second kind [3] where the parameter I appears on both sides of the equation. The solution to this equation may be found by repeatedly substituting the equation into itself, until the required accuracy is found. Each time a substitution is made, it is analagous to following the next ray in the path. The solution will then have the form:

$$
\begin{aligned}
I(x, x') \quad = \quad & g(x, x')[\\
& \epsilon(x, x') + \\
& \int_S \rho(x, x', x'') g(x', x'') \epsilon(x', x'') dx'' + \\
& \int_S \rho(x', x'', x''') g(x', x'') g(x'', x''') \epsilon(x'', x''') \rho(x, x', x'') dx'' dx''' + \cdots \\
&]
\end{aligned}
$$

The meaning of this equation can be easily explained in terms of tracing techniques. If this equation were to be calculated precisely, then as with ray tracing a light ray would be followed from a viewing point until it intersects with some object in a scene. Then at this intersection point, the light incident upon it from all directions would be used to calculate the amount of light transmitted in the viewing direction. Unlike ray tracing and radiosity, all three types of light transport are taken into account, that is diffuse, reflected, and transmitted. Also when the next step in the ray tree is taken, all types of rays must be followed in order to account for all the indirect lighting in the scene. This, as one might expect will take a long time to compute, as it was this type of problem which was encountered by Whitted [51].

2.6.2 Approximate Solutions to the Rendering Equation

The nature of the rendering equation is such that the solution to it encompasses both the techniques of ray tracing and radiosity, in so much that both methods are an approximate solution. Radiosity, which considers only the diffuse light

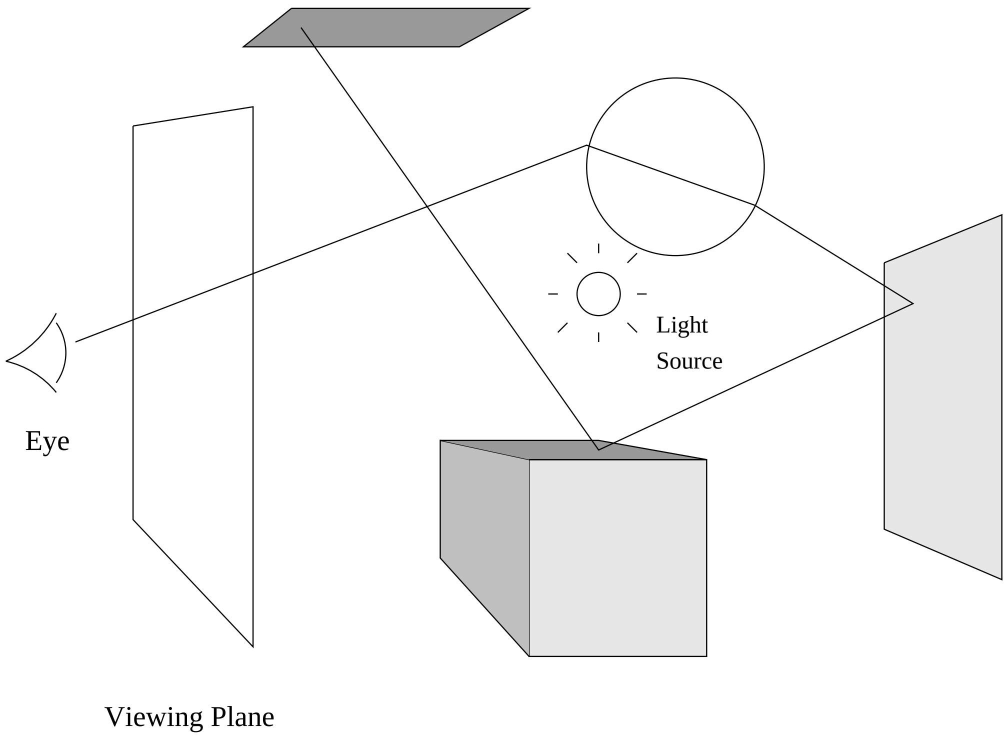


Figure 2.11: Path Tracing

transport, and ray tracing which concentrates on reflection and refraction. As was mentioned above, to solve the equation precisely is an impracticle proposal.

Kajiya [29] noted that one of the most popular methods of solving integral equations was to use a Markov Chain, and that the use of Monte Carlo Markov chains for modelling radiative heat transfer had been in use for quite a while [44]. He then went on to propose a solution to the rendering equation which involved setting up a Markov chain through the scene. This in effect implies following a ray through a scene, but unlike ray tracing, at each intersection point only one ray is followed, creating a *path*. Figure 2.11 shows a simple example to demonstrate this. The type of ray to be followed (i.e. diffuse, specular) is determined by some random factor, as is the direction which is to be followed.

2.6.3 Which Type of Ray to Follow

At each intersection point only one type of ray is followed in the path, which can either be diffuse, reflected, or transmitted. Since we are using Monte Carlo methods we could simply choose some random number in a given range, and make it equally possible for each type of ray to be chosen. It would however be a better solution if the ray chosen reflected the type of surface being intersected. This is sensible because if for example a surface is mostly diffuse, and only a little specular, as many surfaces are, then shooting more diffuse rays than specular would perform a kind of importance sampling. The only guide to the surface properties is from the diffuse, specular, and transmitted reflection coefficients. It would be difficult to base the decision on the wavelength dependent components of each, so instead, inside the data structure for each object, an average coefficient is held. This average coefficient is such that the sum of the average diffuse dc, reflected sc, and transmitted tc values is equal to 1:

$$dc + sc + tc = 1$$

The choice is then made as follows:

- Choose a random number R between 0 and 1.
- If $R < dc$ then shoot a diffuse ray.
- Else if $R < dc + sc$ shoot a specular ray.
- Else shoot a transmitted ray.

In this way, if for example an object only exhibits specular and transmissive properties, then dc would equal zero, and no rays of this type would be followed.

How to best sample the ray directions will be discussed in the next chapter.

2.7 A Comparison of Methods

All of the methods described above have their merits depending on what type of application they are to be used within. A discussion of comparative speeds will not be given, as this is dependent on the hardware used, as well as the number of speed up techniques employed. Beam tracing [25] works almost exclusively for polygonal scenes, and then also most efficiently when only specular effects are modelled. This is then a restrictive method in that curved surfaces cannot be reasonably accounted for. Radiosity [21], as with beam tracing, is used mainly for polygonal scenes, but models the diffuse inter-object reflection, and the solution does not incorporate any specular effects. It models the diffuse reflections very accurately, although the amount of work required for extremely complex scenes soon becomes significant. As has already been mentioned, ray tracing is generally used to model specular effects, as to include diffuse effects is not practicle [51]. An advantage of ray tracing is that it is capable of modelling very complex surfaces, such as Non Uniform Rational B-Splines (NURBS) [40]. It has become increasingly popular to include ray tracing solutions to the radiosity solution in order to model specular effects.

The application of Monte Carlo methods to all types of rendering has become popular [36] to control many aspects of image generation. Path tracing uses Monte Carlo techniques to bring together the best aspects of both ray tracing and radiosity. By introducing a random element into the ray tracing algorithm, diffuse effects may be modelled in a less computationally expensive way. The effects created by path tracing can certainly be compared with images produced using radiosity/ray tracing methods, although the rendering times can be high. Particle tracing attempts to model path tracing in reverse by following particles from the light source until they are incident upon a viewing plane. This produces images of a similar quality to path tracing with the advantage of participating media being included.

Chapter 3

Sampling Techniques

One of the most important objectives in rendering an image, is to ensure that the final colour of a pixel correctly represents the objects that contributed to it [11]. If for example, only one ray is cast for each pixel, then especially around object edges, or in scenes containing many small objects, the final colour of the pixel may be poorly represented. This can be seen in figure 3.1, where the ray through the pixel misses some of the objects, and they are not represented. When this happens, the image is said to exhibit *aliasing*. The solution to this is to cast more rays per pixel in such a way that the a proper account is taken of all objects visible through it. This is called *anti-aliasing*.

3.1 Pixel Sampling Methods

In this section some of the methods that can be used to sample a pixel are discussed with particular attention paid to those which could be used within a

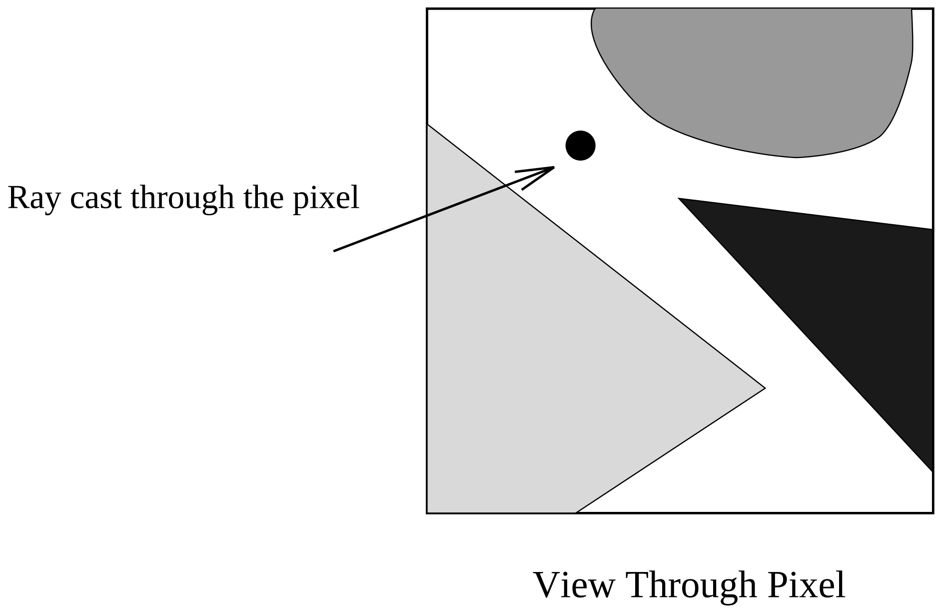


Figure 3.1: Aliasing

path tracing algorithm.

3.1.1 Uniform Supersampling

The simplest way in which a pixel can be sampled is by uniformly distributing the ray positions across the image plane for that pixel [19], as shown in figure 3.2. The rays may be cast through the centre or the corners of the pixel. This more accurately samples the objects visible through a pixel than the straightforward one ray per pixel. However some object shapes may lie in regions which are not sampled by the grid, and be ignored altogether. Also, so called *jaggies* may occur along object edges or intersections, especially those which are regularly shaped. Irregular shapes are by their nature sampled correctly by this method, but since most scenes contain a large number of objects with straight edges, a different method must be considered. A solution to this could be to cast more and more rays until all objects have been sampled sufficiently well. The amount of extra

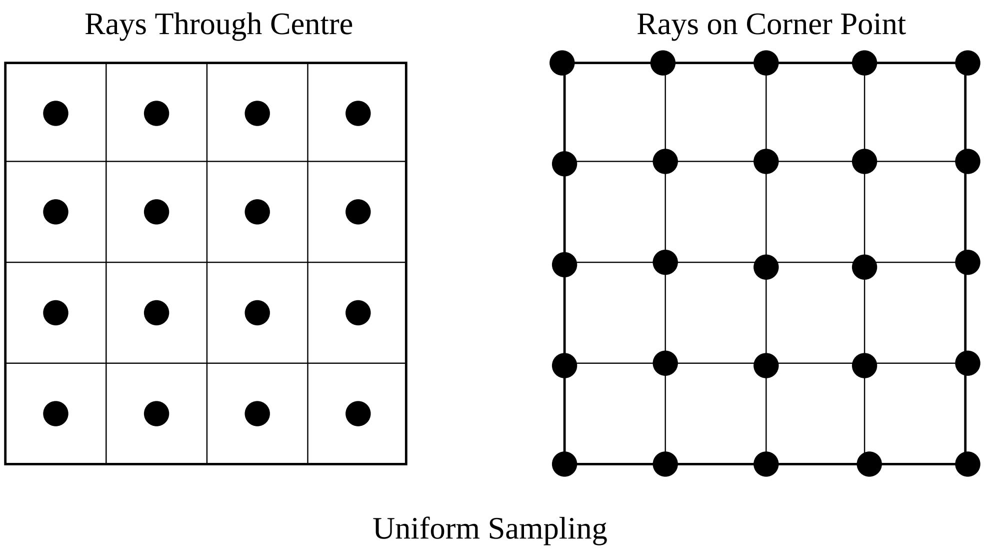


Figure 3.2: Uniform Sampling

work done in this way becomes unreasonable, and for uniform spreads there is usually some object shape which will be badly sampled.

3.1.2 Stochastic Sampling

Stochastic sampling (or *jittering*) [13], [33],[8], [29], is a technique whereby noise is added to a point on the image plane through which the ray is to be traced. When applied across the entire image plane, the resulting jittered ray spread is random, figure 3.3. If a few rays are cast per pixel some objects may not still be very well sampled, but as the number of rays cast is increased, the number of extra rays needed to be cast through that pixel will not be as great as for uniform sampling [13]. Jittering may be applied to the sampling of a pixel or (*domain*) in the image plane in a variety of ways.

1. Random SuperSampling

   This involves taking the whole domain of the pixel on the image plane

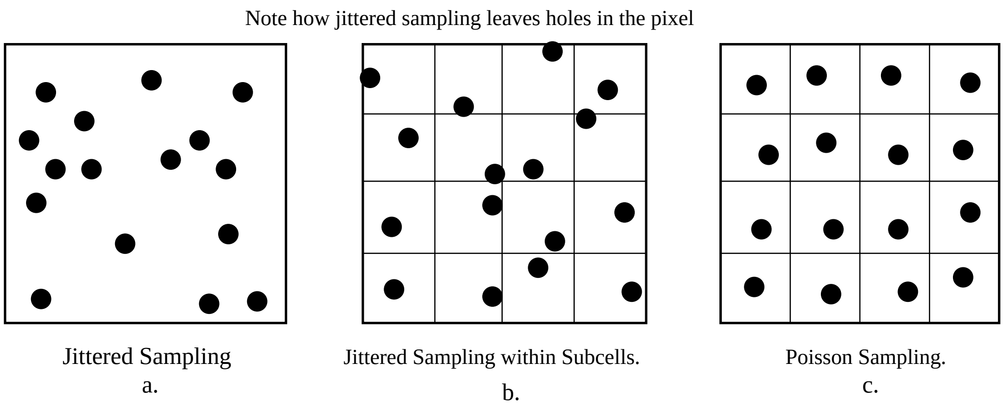


Figure 3.3: Random Distribution Sampling

and randomly choosing a number of rays through it [19], figure 3.3a. This method can result in concentrations of ray points in certain parts of the domain, and this *clumping* can result in areas being badly sampled, especially if few rays per pixel are traced. The returned colour for the pixel can be found by calculating the mean value of all rays cast, since no single ray can be classed as contributing more to the colour than another.

2. Random Stratified Supersampling

   A better way to proceed is to split the domain into a number of subcells, and to choose rays randomly inside those subcells [29], as can be seen in figure 3.3b. This achieves a better spread of jittered rays across the domain, and avoids the amount of clumping which can occur in random sampling. The splitting of the domain is uniform, and so the final pixel colour may again be found by taking the mean path colour.

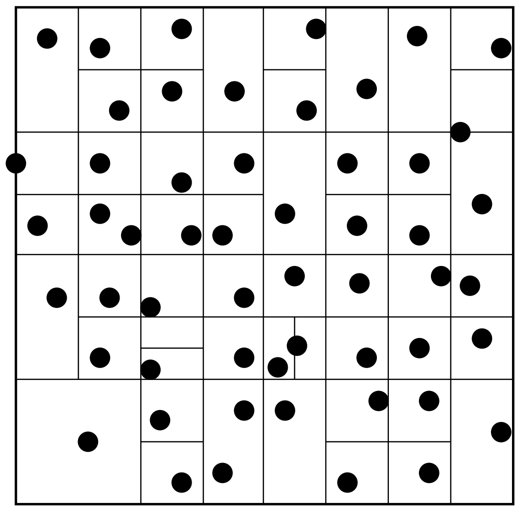
3. Poisson Sampling

   Poisson sampling [9] is very similar to stratified sampling, in that the domain is split uniformly into a number of subcells. The difference is that when a random point is chosen within each subcell, a constraint is placed on how close together any two adjacent points may be, figure 3.3c. This minimum distance restriction controls the amount of noise that can be added to a point, and eliminates clumping effects which may have occurred in stratified sampling. Poisson sampling is more difficult to implement at run time than sequential sampling due to the constraints used, and so one way to implement it is to make a look up table during preprocessing.

4. Hierarchical Sampling

   This method [29], shown in figure 3.4, involves first casting a ray randomly into the domain, and then splitting the domain in half along one of the axes (or *hyperplanes*) which is parallel to the image plane forming two subcells. The subcell containing the initial ray is determined, and a ray is cast randomly into the empty one. The next ray cast is found by choosing one of the subcells, but this time the split is made along the other hyperplane, thus splitting it in the opposite sense. Each time the splitting of the subcell is made along the opposite hyperplane to the last split, and its contribution to the total integral is divided in half. At each split a record is kept of the position and area of the subcell, and the colour of the ray traced through it. In order to implement this kind of sampling a k-d tree data structure described by Bentley [5] must be created.

   The amount and depth to which the domain is split is important in deciding how to determine the final pixel intensity. If the tree structure is followed to the same depth uniformly over the domain, then each point will have an equal area of the domain associated with it. In this case the final pixel colour will be the mean of all ray colours inside that pixel. Another way of proceeding, is to randomly choose the subcell to be split by beginning



Hierarchical Subdivision.

Figure 3.4: Hierarchical Sampling

at the top of the tree, then at each subnode a random variable is used to determine which branch of the tree to follow. This is continued until a leaf is found which may then be split. After a number of these random splits the area associated with each point in the domain will be different.

Yakowitz *et al* [14] proposed the solution that when the pixel sampling has been completed, the colour of each path integral is multiplied together with the area of the subcell it occupies. If the initial area of the domain is chosen to be unit, then the final colour will be automatically scaled to the correct value. Using this method Kajiya [29] reported that the convergence of the integral was superior to the conventional Monte Carlo method.

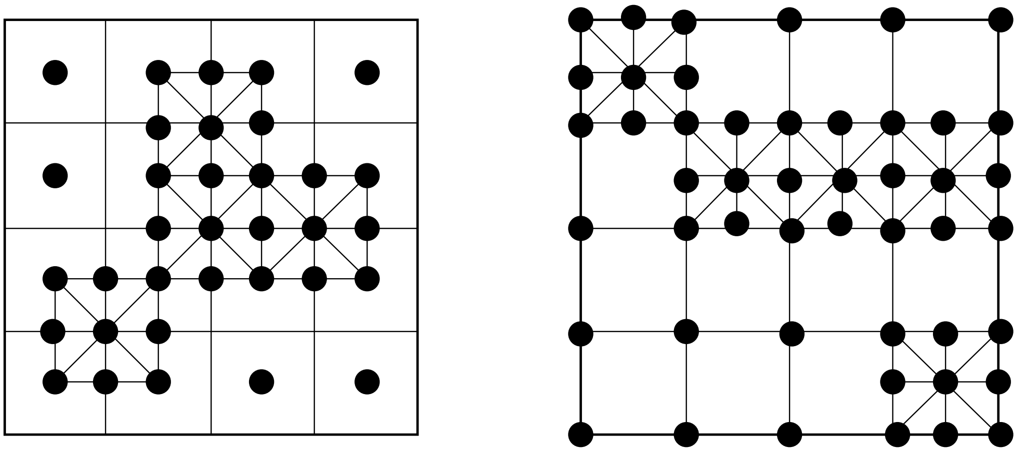
3.1.3 Adaptive Sampling

So far we have talked about the sampling of a pixel in a uniform or random way. This may be adequate for many implementations that are fast by definition, but for images which require a large number of rays cast per pixel, and have extended run times, it is not always sufficient [33]. Some areas of an image may be quite uniform in their colour, and so shooting a large number of rays is not necessary to get the desired picture quality. It would be a good idea to spend more time on the interesting portions of an image, such as object edges and shadow regions. Adaptive sampling works by first casting a certain (maybe set) number of rays per pixel, examining the variance across that pixel, and using it to decide which portion of the image requires extra sampling.

The nature of path tracing makes adaptive sampling difficult, because in comparing two paths in order to see if the variance between them warrants more sampling, they may represent different types of path light. For example one path may have followed diffuse reflection at the first intersection, while the adjacent path may have followed specular reflection. This is not so much of a problem for surfaces which are well lit, because the variance will not be large, however when looking at a surface which is lit by indirect light alone, it may become apparent as each path may pick up colours which vary greatly. For this reason comparing the variance across the whole pixel rather than adjacent rays is a better solution for path tracing.

Adaptive Supersampling

When a number of rays (jittered or not) have been cast in a uniform way, then the variance between ray colours across a domain can be looked at to see if any more attention need be paid to that area. There are two main ways in which the decision to oversample may be determined [33], [19].



Adaptive Uniform Sampling

Figure 3.5: Adaptive Sampling for Uniform Distributions

The first method involves looking at the variance across the whole pixel as opposed to comparing individual path integrals. After a fixed number n of rays has been cast through the pixel, a value such as the standard deviation from the mean is used to calculate how many extra rays to follow. The mean μ_n of n samples is given by:

$$\mu_n = \sum_{i=1}^{n} x_i / n \tag{3.1}$$

where x_i is the ith sample. Using this, the standard deviation about the mean σ can be written as:

$$\sigma = \sqrt[2]{\frac{\sum_{i=1}^{n}(x_i - \mu_n)^2}{n-1}} \equiv \frac{1.25}{n} \sum_{i=1}^{n} |x_i - \mu_n|$$

σ gives the average error of each path from the mean path intensity after n rays. If this is greater than a predetermined threshold then the pixel requires more sampling. The number of extra rays to cast may be based on the standard deviation, depending on how accurate the user wishes the image to be and is discussed later.

Another variant of this method [13], [33] involves doing the above variance check after each ray has been cast, or after every n rays, and continues until an exit condition is satisfied. This is fine for conventional ray tracing, but the nature of path tracing makes this method infeasible. This is again due to the fact that only one type of ray per pixel is followed. As a consequence of this, after n rays have been followed, if for example half were diffuse and half were specular, then if the difference between the intensities of each are great enough, then the exit condition may not be satisfied for an extended time. This is because the standard deviation from the mean may be very large, even though the pixel has been fully sampled. Kajiya [29] reported being unable to find a useful sampling criteria and instead used a fixed number of rays per pixel.

The second way of sampling is to consider a group of four pixels and the area that they surround [19]. If the intensity gradient between any of the four pixels is above a threshold limit, then extra rays are cast as shown in figure 3.5. Attention is then focussed on the smaller groups of four created by this. This is not a good method inside a path tracer because as before the path integrals that will be compared may not be of the same ray type, such as diffuse with specular, and so this technique will not be discussed in detail.

Adaptive Hierarchical Supersampling

Adaptive sampling may be added to this technique as follows. A traversal of the k-d tree is performed after each ray has been cast as normal, but at each subnode a decision is made as to which branch to follow. The criteria used to decide which subnode to split may be one of the following:

1. Uniform sampling, which has already been mentioned.
2. At each subnode choose a branch at random, and continue in this way until a leaf is reached.
3. At each subnode, calculate the difference between the two path integrals, and use this to determine whether or not to split.
4. Use a history of the activity of change within the domain to determine the extent to which the it should be subdivided.
5. Calculate functions during preprocessing to predict in advance where large sources of illumination will be.

Method 4 above has been used in the implementation due to the difficulty in determining a splitting criteria for individual subnodes. A maximum number of rays N is specified by the user, then after 16 rays have been cast through the pixel this number is scaled to a new value N_{new} using the formula:

$$N_{new} = kN\frac{\sigma}{\mu}$$

which was devised by the author, where k is some constant dependent on how accurate an image is required, and σ is the standard deviation from the mean. By using this equation many rays are cast towards interesting portions of the image, whilst more uniform sections receive no more sampling. Although this appears to be somewhat coarse, let us examine its implications a little closer.

In an image, the sections most prone to variance are those which are not very well lit. This is because the brighter a surface is, the less noticeable any variance across it becomes, whereas for dark surfaces, any variance will show up clearly as speckling. Considering the above formula for these two cases we get:

- For a given σ, the percentage of extra rays cast for a bright surface, which will have a high mean, will be low because σ will be small in comparison to the mean.

- Badly lit surfaces will have a low mean intensity, however the ratio of σ to to the mean will be high giving a large percentage of the maximum rays being followed.

The consequence of this is that dark regions are sampled very heavily compared to bright regions, which is what is desired. Also, k can be chosen in such a way that the number of extra rays is actually larger than the maximum N, meaning that surfaces with a high variance are sampled even heavier.

3.2 Sampling of Ray Distributions

The distribution of higher generation rays has been well researched as there is a need for accurate sampling of these rays in all types of global illumination algorithms. As for pixel sampling the distributions fall into the categories of random, uniform, and adaptive, and one of these must be chosen for each of the ray types. In order to model random behaviour the following equation is required, described by Lange [32]. It is the cumulative distribution function [1]:

$$R = P(\xi) = \frac{\int_l^{\xi} f(x)dx}{\int_l^h f(x)dx} \tag{3.2}$$

describing the transformation of a set of uniformly distributed random numbers R in the unit interval (0,1), into a set of usually non uniformly distributed random numbers ξ in an interval (h,l). $f(x)$ is the probability density function, that is the average distribution that must be satisfied by the ξ values determined by the simulation scheme, or simply the proportion of values that lie in the region $d\xi$ around ξ. Each value ξ is then obtained by choosing R at random, and using the function relation $P(\xi)$ to determine the corresponding value of ξ. By choosing a suitable function $f(x)$ we achieve a description of the desired physical process.

3.2.1 Ray Distributions for a Reflecting Surface

In ray tracing, the technique of distributing rays about the specular direction to produce blurred or hazy reflections has been well documented and is discussed by Cook *et al* [10]. When light is scattered by a surface, the ray directions are not only in the perfect specular direction, but are distributed about that direction as shown by figure 3.6. The way in which these rays are distributed can be thought of as random, which makes it suitable for Monte Carlo methods.

The main problem is to calculate the size of the solid angle in which the rays should be distributed. The method originally suggested by Whitted [51], was to use the same distribution function as determines the highlights, although he did not specify any particular function. We are using the Phong model [39] to represent the highlights, so a method of distributing rays according to this may be used.

Because the Phong model produces circular specular highlights, we can choose the circumferential angle ϕ which has range $(0,2\pi)$, by taking a random number R between 0 and 1, and multiplying it by 2π such that $\phi = 2\pi R$.

The cone angle between the reflected direction and the random ray will lie in the

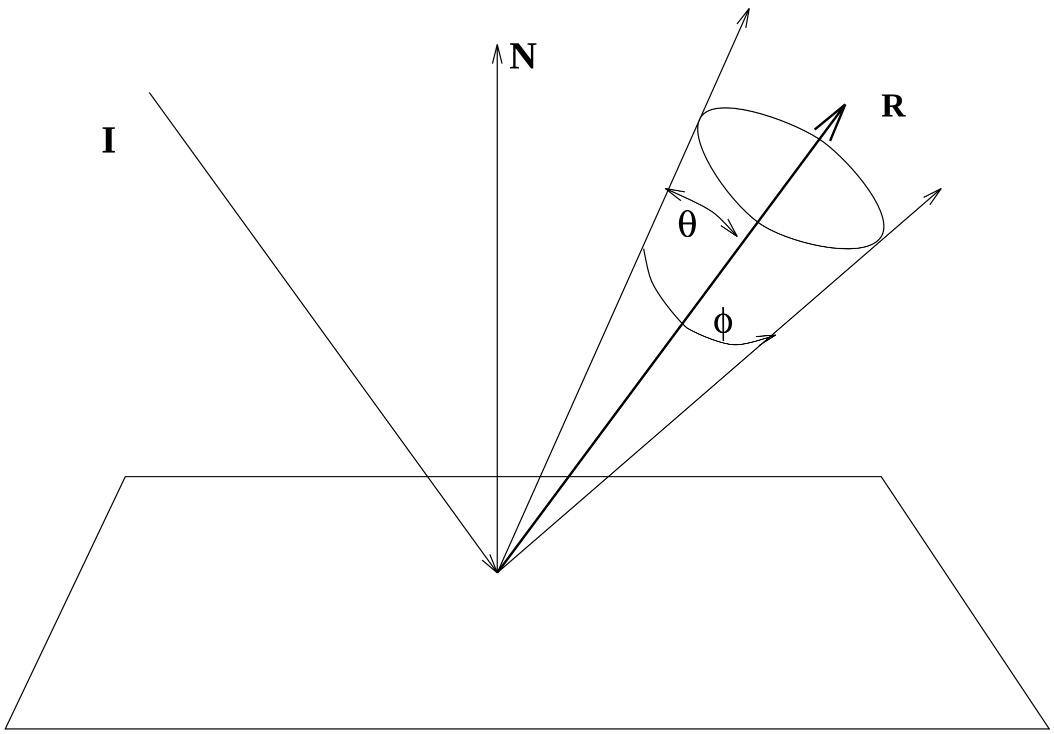


Figure 3.6: Ray Distribution for a Reflecting Surface

range $(0,\pi/2)$, but will be more concentrated at smaller angles. The equation for calculating this can be found by inserting the intensity function for specular reflection into equation 3.2 above:

$$R = P(\theta) = \frac{\int_0^{2\pi}\int_0^{\theta} I\cos^n\theta\sin\theta d\theta d\phi}{\int_0^{2\pi}\int_0^{\pi/2} I\cos^n\theta\sin\theta d\theta d\phi} = 1 - \cos^{n+1}\theta$$

therefore:

$$\theta = \arccos(\sqrt[n+1]{(1-R)})$$

This distribution function samples mostly around the specular direction, which is reasonable as light at greater angles gives little contribution to the final intensity. Also, now that the cone of angles has been sampled, the Phong term (section 2.2.2) may be removed when calculating the reflected intensity, and only the specular reflection coefficient need be considered, thus speeding up calculation.

The above method has been used inside the implementation although some consideration of other methods should be made. Consider the situations which might occur. If a surface were very shiny then random sampling would suffice, as the solid angle in which the rays would be distributed would be very small. For rough surfaces however the solid angle will be fairly large. In this case, random sampling may not be sufficient unless a large number of rays are followed. Uniform sampling would certainly reduce variance, although adaptive sampling may be unnecessary as we are attempting to produce a hazy reflection, and the extra work would be wasted. It has been chosen to use randomly distributed rays for the sampling, but to control the number of rays cast depending on the size of the solid angle.

One last mention must be made concerning rays which only graze the surface of an object. In this case some of the distributed rays may actually point below the surface, and as such can be discounted. It is easier to discount these rays than calculate a special function for small incident angles.

3.2.2 Ray Distributions for a Transmitting Surface

The distribution for this type of surface is almost exactly the same as for a reflecting surface in that rays are distributed around the transmitted direction according to the Phong exponent, see figure 3.7. Again the Phong term may be omitted in the intensity calculations.

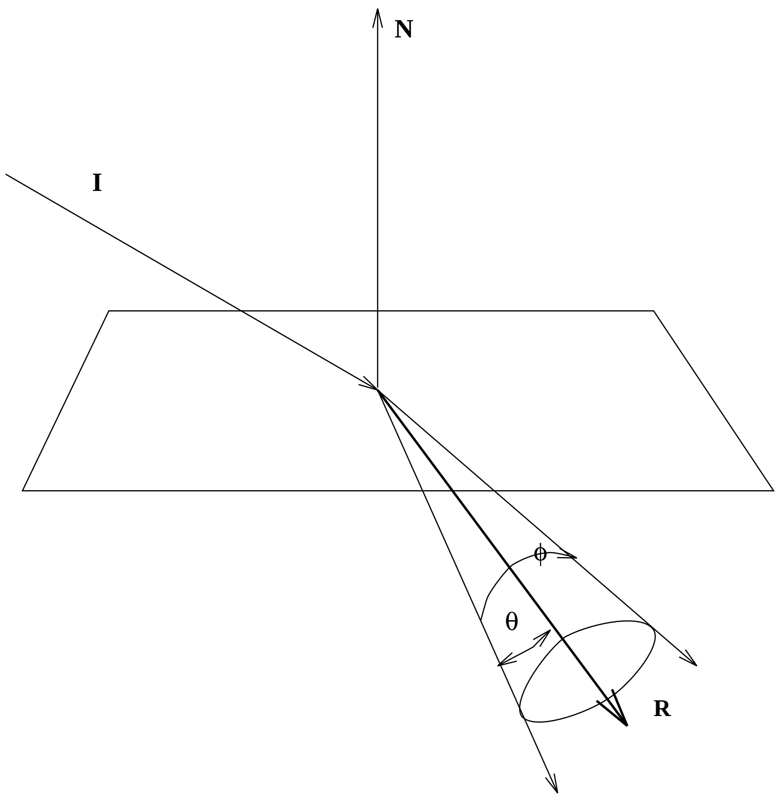


Figure 3.7: Ray Distribution for a Transmitting Surface

3.2.3 Number of Specular Rays to be Followed

Now that we know how to distribute the specular rays, the question of how many to follow in order to calculate a reasonable intensity must be discussed. Lange [32] suggests taking a maximum number of rays N, and following a percentage of this number dependent on the specular reflection coefficients k_{sr} and k_{st}. This may suffice in a simple scene but would become inadequate for a complex one. If for example, a surface has a low reflection coefficient and a low Phong exponent, then this model will cast very few rays even though there may be a source of large illumination in the solid angle of directions which may consequently be badly sampled. The reverse case is when a surface is very shiny and very reflective. In this case many rays would be followed into a very small solid angle, oversampling the area.

Due to the apparent inadequacy of this model the number of rays to be followed will be based upon the Phong exponent n, as opposed to the specular coefficients. In this way for small solid angles, fewer rays will be followed, but for larger angles the number of rays will be high. This also performs a kind of importance sampling. The method takes a maximum number of rays N, and scales them according to the equation:

$$N_{new} = N/(\log_{10} n)$$

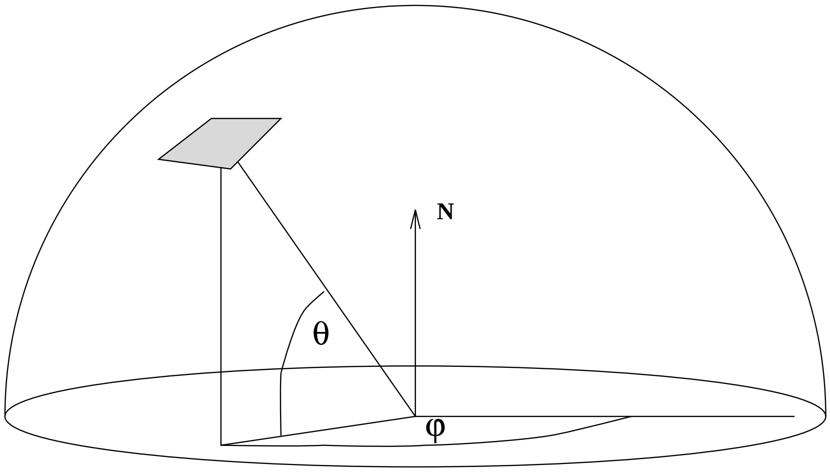
This equation, devised by the author, gives a reasonable spread of values for the whole range of the Phong exponent. Other functions such as roots were considered but the log function appeared to give the best approximation. This quantity may be precalculated.

3.3 Ray Distributions for a Diffuse Surface

For a surface exhibiting diffuse reflection the rays incident from the hemisphere of directions about the normal contribute to the emitted intensity. There are several ways in which the total diffuse contribution can be calculated for use in path tracing. As with the specular distributions, these can either be random, uniform, or adaptive.

3.3.1 Random Distribution

Lange [32] chose to randomly distribute the rays over the hemisphere of directions, but with the distribution chosen so that more rays are cast for angles of less



Diffuse Rays Cast Over the Hemisphere.

Figure 3.8: Random Diffuse Rays

than 60^o between normal and ray. This was done because the cos term used to calculate intensity implies that these rays make a more significant contribution to the intensity incident upon that point. A random ray on the hemisphere may be calculated in the following way. As for the specular rays the circumferential angle ϕ is found by multiplying a random number in the range (0, 1) by 2π such that $\phi = 2\pi R$. Calculation of the cone angle is done similarly to the specular ray in that the intensity function for diffuse transport is substituted into equation 3.2:

$$R = P(\theta) = \frac{\int_0^{2\pi} \int_0^{\theta} I \cos\theta \sin\theta d\theta d\phi}{\int_0^{2\pi} \int_0^{\pi/2} I \cos\theta \sin\theta d\theta d\phi} = \sin^2\theta$$

so that:

$$\theta = \arccos(\sqrt{}(1 - R)) = \arccos(\sqrt{}R)$$

this is true because R is in the range (0,1), therefore $1 - R$ is also in that range.

As for specular rays, Lange [32] chose to base the number of rays randomly followed on the diffuse reflection coefficient. As we are sampling over the whole hemisphere of directions, then in only relatively simple scenes, large amounts of

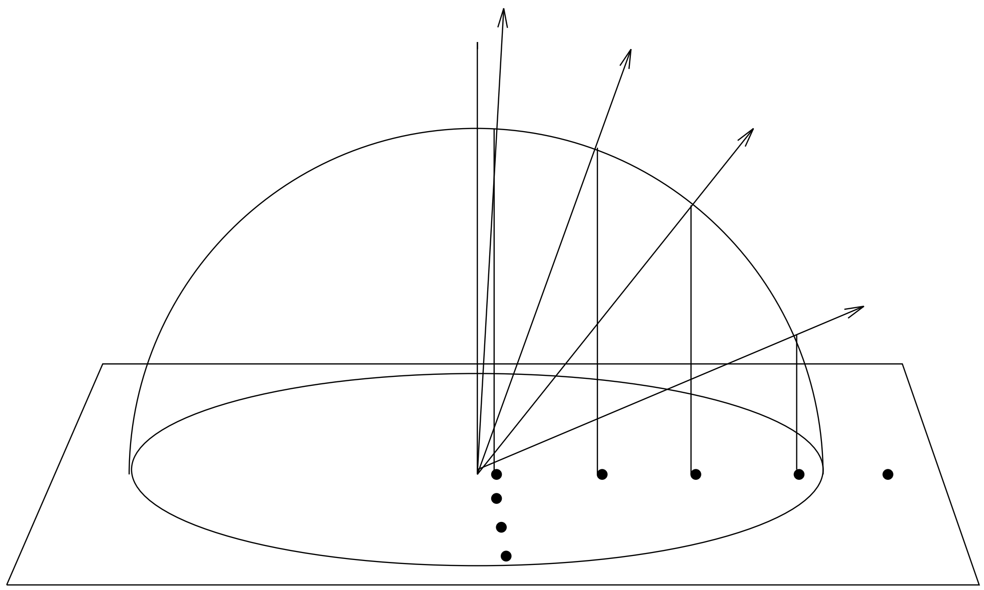


Figure 3.9: Uniform Diffuse Rays

illumination could very easily be missed. Although the diffuse coefficient may only be small, if the surface at that point is well illuminated by other surfaces, then this model will be insufficient unless the maximum number of rays followed is high due to aliasing effects. This method has been employed to produce the images in chapter 6 although it was only sufficient in reasonably well lit scenes. The other sampling methods now described must be considered for badly lit scenes.

3.3.2 Uniform Distribution

If we are to represent all of the light incident on a point, then because the solid angle of the hemisphere is so large, it might be better to distribute the rays uniformly so that holes which will appear in the random choice do not occur. This is slightly contrary to the view held for sampling of a pixel domain, however the consequences of leaving gaps over the hemisphere is of greater significance.

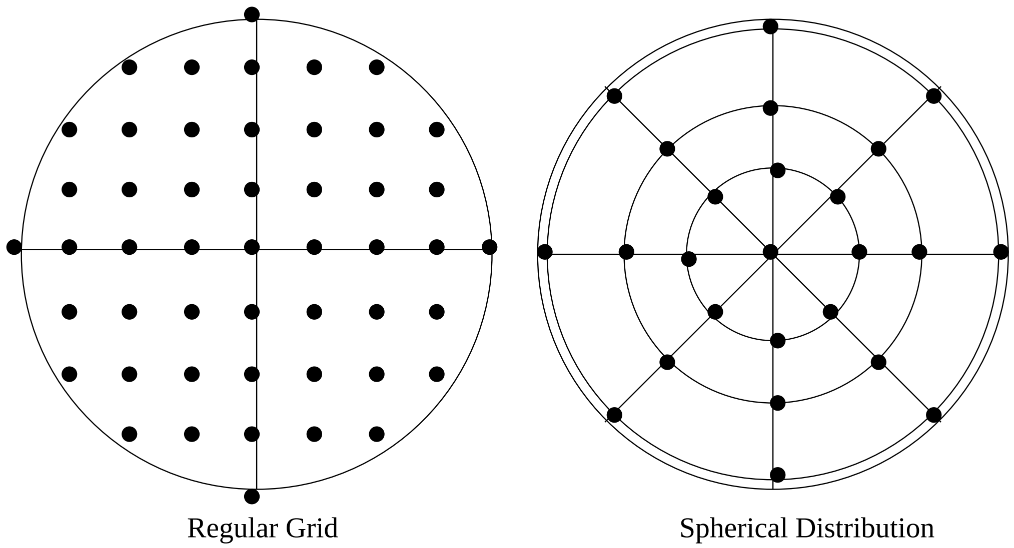


Figure 3.10: Distribution of Points Across Plane of Hemisphere

An easy way of producing a spread across the hemisphere would be to project points on the plane of the surface up onto the hemisphere, and use these points to follow the paths from, as in figure 3.9. The points can be distributed regularly on a grid, or spherically across the plane, figure 3.10. This nature of this method means that it will cast more rays at small cone angles than large ones as was the criteria for randomly distributing rays. As was done with the uniform distributions for the pixel, the points can be jittered on the plane before they are projected, in order to reduce aliasing effects. The diffuse contribution using this method can be calculated by taking the mean of all paths followed.

Another way of distributing the rays is to take points on the hemisphere at every θ degrees on the cone angle, and every ϕ degrees on the circumferential angle, which will cover the entire hemisphere uniformly. In this way however there is no bias towards the top of the hemisphere, and as such oversampling will occur. A form of jitter can be introduced by adding noise to the θ and ϕ angles calculated.

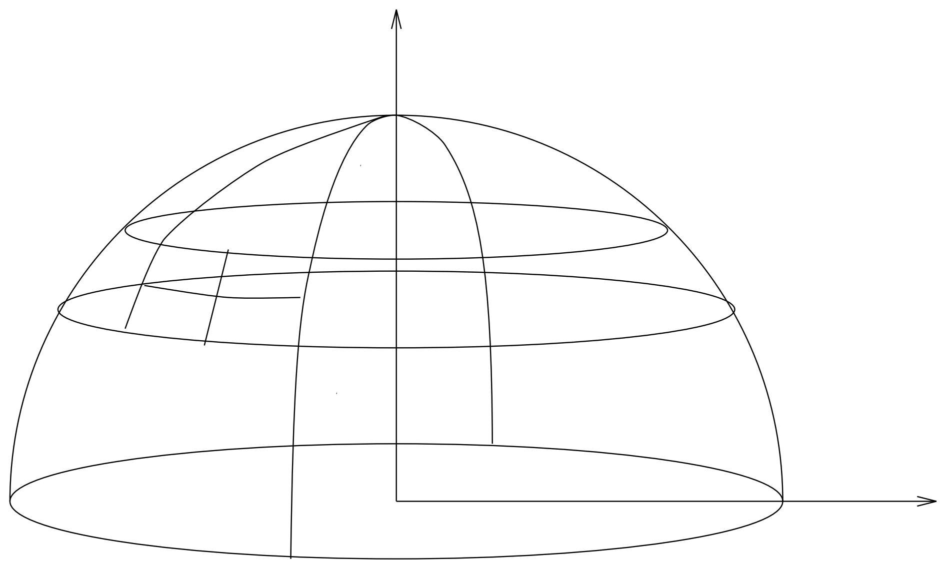


Figure 3.11: Multidimensional Splitting of a Hemisphere

3.3.3 Multidimensional Splitting

Kajiya [29] employed a method based on the hierarchical splitting already described, although he gives no indication as to how many rays were cast over the hemisphere. This method considers three dimensions instead of the two used in pixel sampling. Initially a ray is cast randomly through the hemisphere, and then a split is made along one hyperplane, a sample is then cast into the empty half. All subsequent splits alternate between the three hyperplanes, this is illustrated in figure 3.11. As with pixel sampling, every time a subcell is split, the contribution from the path integral associated with it is divided by two. A traversal of all subcells must be made after sampling, and the path integrals multiplied with the associated subcell size. Sampling in this way causes the variance to converge faster than for the uniform method [14].

3.3.4 Adaptively Sampling the Hemisphere of Diffuse Directions

Both Lange [32] and Kajiya [29] produced images which where well lit and where the indirect illumination only coloured the already well lit surfaces. For random sampling it has been found that in order to reduce variance across an image to a tolerable level, many diffuse rays must be cast per intersection both for random and uniform sampling. This is especially true for complex images where many surfaces are lit only by indirect illumination. One drawback to random sampling is that there is no guide as to how much each path integral contributes to the final intensity. If uniform or hierarchical sampling is used then this is overcome, however these methods become very computationally expensive if the hemisphere is to be sampled accurately. Some areas of a scene may be very uniform in colour, but many rays used to sample them.

A great deal of research has gone into the adaptive sampling of a hemisphere in the field of radiosity [41],[45], [46], [17], and so this is where we shall look to find a technique for modelling the diffuse distribution. Many of the methods currently employed are not applicable to ray tracing, as radiosity is more concerned with patch visibility than ray tracing. Although any effort to model accurately the diffuse inter-object reflections between surfaces is inherently computationally expensive, as described by Whitted [51], for complex scenes Pattannaik [37] suggests that it becomes imperative in his discussion of Monte Carlo methods. The methods already mentioned involving projecting rays from the surface plane to the hemisphere can be modified for adaptive sampling.

Kirk and Arvo [31] state that a feature common to all multi- stage Monte Carlo integration techniques is the notion of using a small number of samples as a first stage to detect regions where additional sampling is required in order to achieve a reliable answer, that is, one with an acceptably low level of noise. Also that a decision to invoke a more costly method as a second stage is based upon a statistic they call the *variation* which is a function of the first stage. This is the kind of sampling, which is *statistically biased* [27], that we will consider. Kirk and Arvo continue to state that the expected value of a n statistically biased samples will be given by:

$$E[\eta] = I + \sum_{i=1}^{k} \omega_i^n (I_i - I)$$

where I is the true mean, I_i is the ith sample, and ω_i is the probability of the ith sample. The summation on the right of this equation is the amount of bias which decreases as the number of initial samples is increased. This may be taken into consideration for the initial ray distributions.

The other general method involves taking a small set of initial estimates and using the *variation* of all of these as a base for sampling the entire region to a greater extent, thus no specific areas are subject to further sampling, It is this kind of sampling that was used for the pixel, and is said to be *statistically unbiased*. The amount of extra sampling done in this way may exceed the amount done using the previous method.

Determining a criteria for further sampling is not easy. Parameters which must be taken into account are the intensity of the path integral, the distance from the surface from which it was calculated, and the angle between ray and surface normal. The greater the angle between normal and ray, the less it contributes to the overall path intensity. If two samples are being compared and both are found to be of equal intensity, then it could be said that no further sampling need be considered. If however the two surfaces from which the samples originated differ greatly in distance, then in fact the emitted intensity from each surface will vary significantly, and this should be taken into consideration.

It has been chosen to distribute the rays across the hemisphere spherically about the normal as in figure 3.10, but with an amount of jitter added to each point proportional to the distance from the centre of the hemisphere. If rays are initially spread in this way then a comparison of adjacent rays to determine changes in the intensity gradient may be carried out. If a predetermined criteria for further sampling is satisfied, then further points on the hemisphere may be chosen as indicated by figure 3.12. Earlier it was suggested that a direct comparison of path integrals was an infeasible proposal, however in this case we are more interested in directions of intensity change, rather than an accurate comparison of intensities.

A criteria for further sampling may be determined in the following way. Initially the mean μ_n is calculated for four adjacent path integrals using equation 3.1. By consideration of this mean the path integrals may be compared and checked against each other for variance. If the following inequality defined by the author is true for any two path integrals, then the enclosed area should be further sampled:

$$I_{max} - I_{min} > \alpha k \mu_n$$

where I_{max} and I_{min} are the maximum and minimum intensities of the path integrals, α is a variable dependent on the angle of the between the paths and the surface normal, and k is a user defined constant dependent on how sensitive the adaptive criteria is to be. This equation thus takes into consideration angular

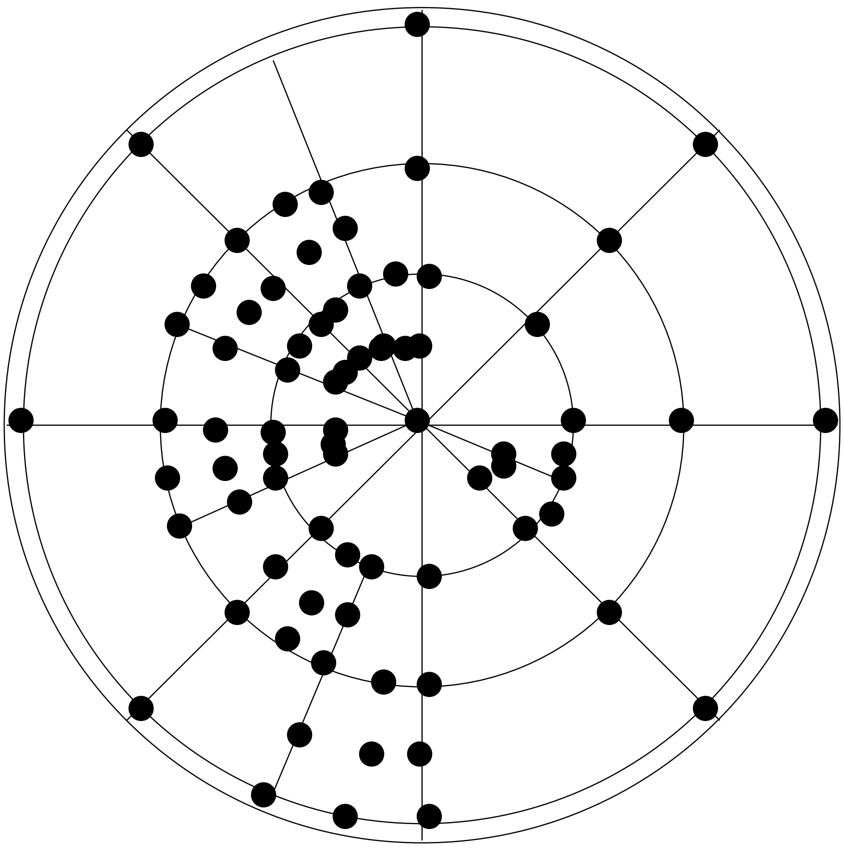


Figure 3.12: Adaptive Sampling of the Hemisphere

dependencies, but discounts variation in distance. An investigation into suitable values for these constants has not yet been possible. A last criteria that should be mentioned is that for surfaces with a very low diffuse reflectance coefficient, it may be possible to ignore the diffuse component entirely.

Precalculation of Ray Distributions

Calculating and choosing further ray directions at run time is a computationally expensive task, especially as it must be carried out for every first generation intersection. If ray directions were chosen during preprocessing and applied in some way, then the run time would be reduced.

For a given circumferential and cone angle, θ and ϕ, about a surface normal, the way in which the ray direction is calculated is to first find two orthogonal unit vectors which are tangential to the plane of the surface. These vectors are scaled according to the values of θ and ϕ and added to the normal. A sensible value to store during preprocessing then would be the scaling values for a large subset of the rays across the hemisphere. This creates another problem in that there are only one set of directions to be applied for each intersection. If adjacent intersection points have the same two plane vectors then they will be sampling exactly the same positions in the scene. The way to get around this is by rotating the calculated plane vectors by a random amount for every intersection. In this way the hemisphere of precalculated directions is automatically rotated. In an initial implementation, 1536 ray directions have been precomputed. If any further accuracy is required then a simple routine is included to choose points randomly within the particular region under scrutiny. Time restrictions have precluded a full implementation of adaptive sampling.

Chapter 4

Implementation Aspects

Now that the sampling techniques have been covered fully, the other aspects of implementation may be discussed. One of the greatest problems facing an implementation is the time taken to render an image. If a long time were available and the best methods were used then very realistic images could be created, however the times involved would be excessive. A compromise must then be found between the time taken to render, and the quality of the image required.

4.1 Intersections and Scenes

One of the most important aspects of implementing a ray tracer is to have fast intersection algorithms. Rubin and Whitted [42] showed that their program spent most of its time computing ray-object intersections. It is then, very important to use highly efficient ray intersection routines. Glassner [19] in his book brings together most of the faster intersection routines for primitive objects. Some of these are outlined in appendix A. Complex objects may be formed using combinations of the primitive objects described.

4.1.1 Describing the Scene

When we are trying to create a scene and store it inside a data structure, the positions and sizes of objects should be described in a sensible way. It would also be desirable if different implementations could utilise the same scene description, in order that images can be compared. In the Manchester Computing Centre, Computer Graphics Unit, a *scene description language* has been developed by Gatenby [18], which given an object file, written using a defined syntax, parses it and stores all the information internally. An example scene for a simple coffee table may look like:

```
compobj coffee_table

  /* position the whole thing where we want it in the scene */
  shift 10.0 20.0 15.0

  /* The table top */
  cylinder
    0.0 0.0 0.0 capped 0.0 0.0 1.0 capped 1.0 0.0 0.0
    scale 10.0 10.0 0.1 shift 0.0 0.0 10.0
    props drc 0.6745 0.5373 0.4353
  end
```

```
  /* The leg */
  cylinder
    0.0 0.0 0.0 capped 0.0 0.0 0.0 capped 0.0 1.0 1.0
    scale 2.0 2.0 9.8
    shift 0.0 0.0 0.2
  instprops metallic_grey /* defined elsewhere */
  end

  /* The base */
  box
    -2.0 -2.0 0.0   /* bottom corner */
    2.0 2.0 0.2     /* top corner */
    instprops metallic_grey
  end

end /* coffee table */
```

This information is passed to implementation dependent subroutines from which the user can extract whatever information is required for an object, and store it inside their own data structures. The scene description language allows most of the usual primitive object types to be defined, and also combinations of them to form composite objects, and then allows those objects to be transformed and *instantiated* many times in a scene. Object properties and viewing directions may also be included. One of the great advantages of using a scene description language of this type, is that libraries of objects may be built up over time and used to quickly build very complex scenes which would otherwise have taken a long time to create.

4.1.2 Transformation of Primitive Objects

Although it is possible to define objects at any point in space, it is often desirable to define an object such as a sphere at the origin, but then to give it an associated transformation matrix. In this way, many object shapes which would normally be very difficult to include can be easily defined, such as an ellipsoid. If a transformation is applied to objects which are simple to intersect, such as polygons, then the transformation can be applied in preprocessing, and all of the points transformed and stored as if the object had been defined as the transformed object. Another example might be for a sphere which is defined at the origin to be then scaled along one axis, and then intersected as if it were an ellipsoid. If an ellipsoid is then rotated such that its axes are no longer parallel to the world axes, then intersection becomes less easy. A technique frequently used to overcome this [4] is not to attempt to transform the object, but to let the object have an associated transformation matrix, and to transform the ray itself into object space. In order to do this the inverse matrix may be determined during preprocessing. Once the ray has been transformed to object space, the intersection distance can be calculated and transformed back into world coordinates.

4.2 Colour

The goal in rendering an image, is to find the correct colour of the objects seen through the virtual viewing screen. This issue of how to represent the calculated intensities must be addressed.

4.2.1 Ray Wavelengths

In our discussion of light so far, no mention has been made of how the individual wavelength intensities at each intersection are calculated. The equations suggest that the entire spectrum of wavelengths should be considered. This is very impracticle as the colour for an infinite number of wavelengths could then be calculated. Most tracing routines take a small subset of wavelengths. The primary colours red, green and blue, which are are used to produce television images, make them a particularly suitable subset of wavelengths to follow, and in fact that is the case. Meyer [35] suggests using a subset of six or nine wavelengths to improve the quality of the image, but this is not entirely necessary for the purposes of this project, and it also increases the run time. Coefficients of reflection and refraction are used to model the behaviour of light interaction. A consequence of following separate wavelengths, is the fact that when a ray is transmitted through or reflected off an object, the direction of the ray is affected differently for separate wavelengths, the extent to which is governed by the surface properties. Only by following the individual wavelength rays can transmission be modelled correctly. This is an impracticle proposition for a path tracer due to the number of rays that would have to be followed. Only one ray direction is chosen to be followed as an approximation, meaning that the effect of light splitting will not be modelled. The type of scene that a path tracer might be used for would not normally contain any objects of the type which split light in this way, so it is not an unreasonable approximation.

4.2.2 Scaling the Intensities

It is all very well spending vast amounts of time correctly modelling a scene, if at the end of it when the calculated intensities are viewed, no effort is made to make them appear realistic on screen. Converting light intensities into colour values that would seem correct to the eye is a complex matter, and much research has gone into deriving colour models for this purpose [34], [12]. Parameters which would be have to be considered would be the particular phosphur content of the screen used to view, and the wavelength response of the eye. Due to this complexity a simpler approach has been adopted discussed by Glassner [20]. While the light intensities of each pixel are being computed, a record is kept of the greatest and least calculated intensity for all three wavelengths. When the rendering has finished, this intensity is used to scale all of the other intensities to (r,g,b) values B_{rgb}. An (r,g,b) value consists of three integers between 0 and 255 for a 24-bit image (one number for each of the red, green, and blue intensities). The user defines two constants, brightest and dullest, to define the range of the (r,g,b) values and then the intensities can be scaled using the equation.

$$B_{rgb} = dullest + (brightest - dullest)\frac{B_{W/m^2}}{B_{Max}}$$

Where B_{Max} is the maximum pixel intensity (in any colour band), and B_{W/m_2} is a pixel intensity value.

4.2.3 Gamma Correction

When viewing an image which includes a lot of indirect illumination, some of the objects may be appear very badly lit. This is especially true after the scaling described above has been included. In this case the low (r,g,b) values may not be represented on the displaying medium, although they may be important to a realistic looking image. Gamma correction involves scaling the image intensities so that the minimum (r,g,b) values become greater, and the very dim colours are represented, large (r,g,b) values are scaled very little. This can be done at the same time as the intensities are scaled, however most displaying utilities include this option, and so the amount of gamma correction can be controlled using these. The equation governing gamma correction for the intensity I_{rgb} [15] is:

$$I_{rgb} = kV_{gun}^{\gamma}$$

where V is the potential difference across the Cathode Ray Tube (CRT) being used to direct electrons onto the screen phosphurs, and k and γ are both constants that depend on the particular phosphurs used to make up the CRT screen.

4.3 Lighting

So far the illumination discussed has been with regard to individual light rays intersecting with a surface, the origins of these can be thought of as point sources inside a lighting model. The problem with this is that in a real scene there are no actual point sources, as any real light emitter has some finite surface area, and so the model described is inadequate. Shadows caused by point sources are very sharp and do not exhibit the penumbra regions associated with area sources.

In the next few sections different types of light source will be considered, and whether they have a place inside a realistic lighting model.

4.3.1 Shadows

Before we discuss light sources, a brief mention of shadows and shadow testing will be made. When a ray intersects an object, the contribution to the light emitted from that surface due to all the lights in the scene must be calculated. Some of the lights may not be visible to that point however, and so their contribution need not be counted. In order to find out which lights are visible, an intersection test must be carried out using a ray from the light source to the intersection point. If any object occludes that ray with an intersection distance less than that between the ray and the light, then the point is not illuminated by that light source. Difficulties arise when area sources are introduced, because an intersection point may only be illuminated by part of an area source. This causes *penumbra* to be formed at the shadow boundary. It is not in this case completely obvious which rays to perform an intersection test on. Figure 4.1 shows how the different types of shadow are formed on a surface. This issue will be addressed in a later section.

4.3.2 Point Light

A point light is as its name suggests, a light emanating from one point in space which has an infinitesimally small surface area, and as such the shadows produced are very sharp, figure 4.2. In nature, point light sources do not exist, since anything that emits light must have some surface area. Also, there is no way

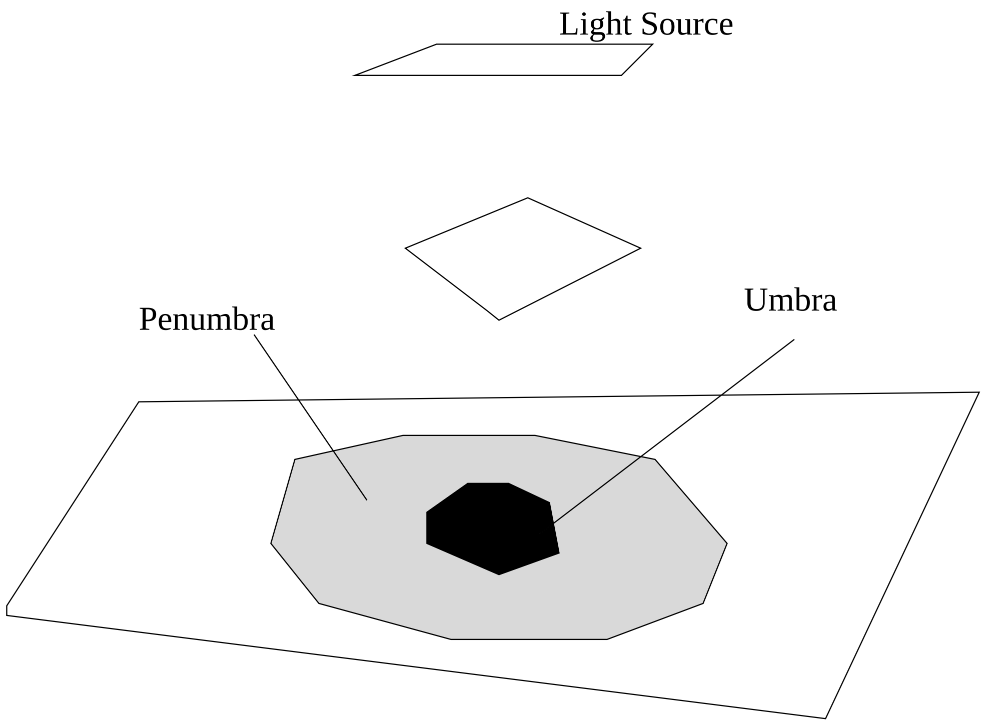


Figure 4.1: Shadow Example

of visualising a source of this type, and only its effect on the environment can be seen. Point light sources then, should not really be used to simulate realistic lighting. They are good for test purposes, or for when realism is not an important issue, such as for geometry viewing.

4.3.3 Directional Light

A directional light source is considered to be a point source at infinity, such that all light rays passing through the scene from that source are parallel, and are in a predetermined direction, figure 4.3. This type of light source could be used to model effects like sunlight. No account can be made of the attenuation with distance, resulting in all objects in the scene being rendered with equal intensity incident upon them. It would be unsuitable to use directional light in an enclosed area, and thus is not really a useful or suitable type of lighting in most cases.

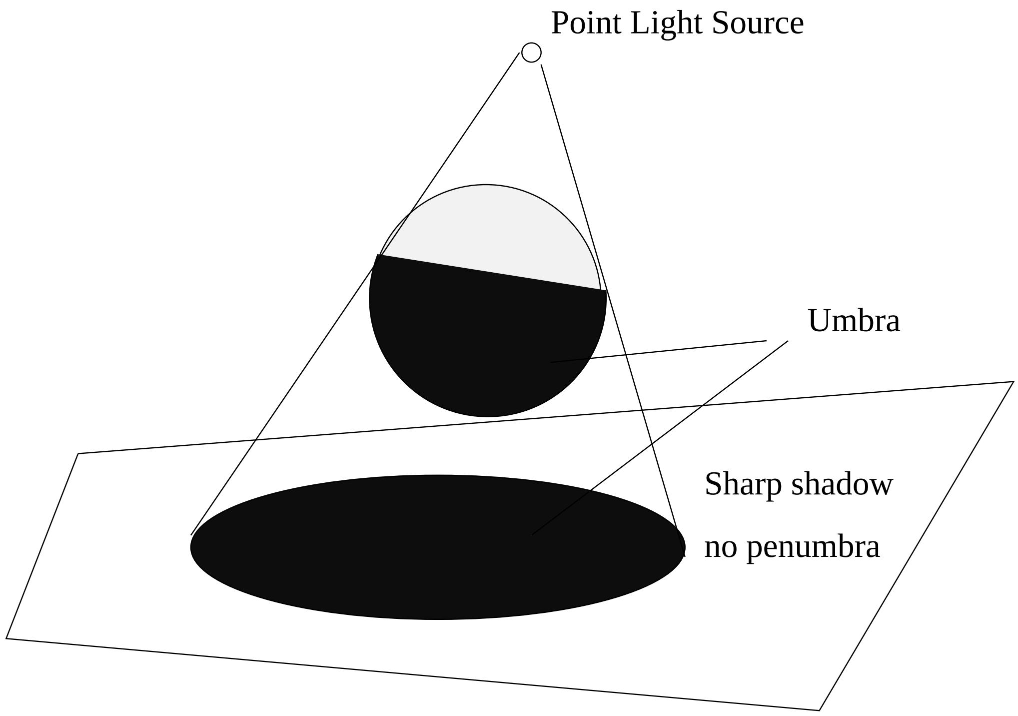


Figure 4.2: Point Light

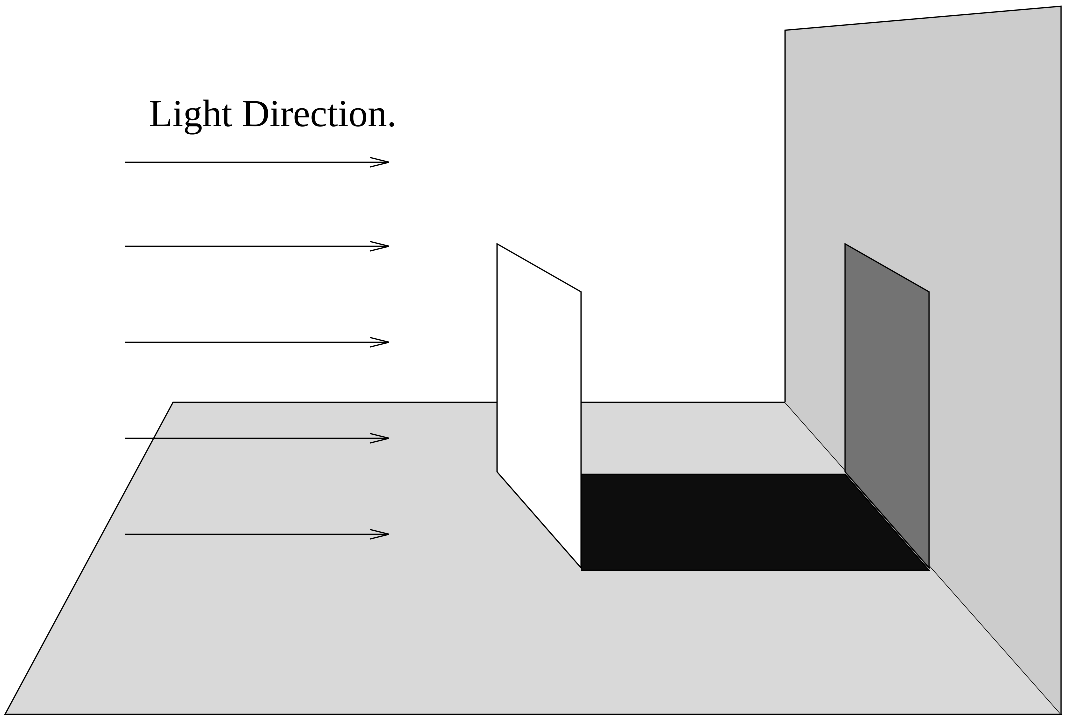


Figure 4.3: Directional Light

4.3.4 Spot Lights

One way of modelling a spot light, proposed by Warn [50], is to consider the light as having an origin **P**, and pointing in some direction **D**. Viewed along **D** the light has some intensity I, but as the viewing point moves away from **D** the intensity drops off according to the Phong exponent [39]. Also, at a certain point, when the angle θ between the viewing direction and the light direction is greater than the *cutoff angle*, then the light is not visible beyond that angle. This model gives a reasonable effect at a low computational cost if the Phong exponent is chosen carefully. One of the problems with this model is that the actual light enclosure cannot be seen, and thus there is no indication as to where the light is coming from, apart from its effect on the surfaces of the scene. Although this model produces a reasonable effect, it assumes that the light is a point source emitting over a certain solid angle. We have already said that point light sources are not a good way of modelling realistic lighting, and so this model must also come under the same criticism. The shadows produced by spot lights are not sharp, as light is reflected from the inside of the lights enclosure. The nature of the Phong model means that we are restricted to circular light sources, although this type of spot light has been included in the implementation.

What is required is for some enclosure to be placed around the light. A cylinder capped at one end could be placed around the model described above, however the problem of light reflecting off the inside will still be apparent. A better solution would be to place a point light source at the rear of the cylinder thus creating an effective cutoff angle dependent on the cylinders design. The Phong term introduced above, which attempted to take into account reflections off the

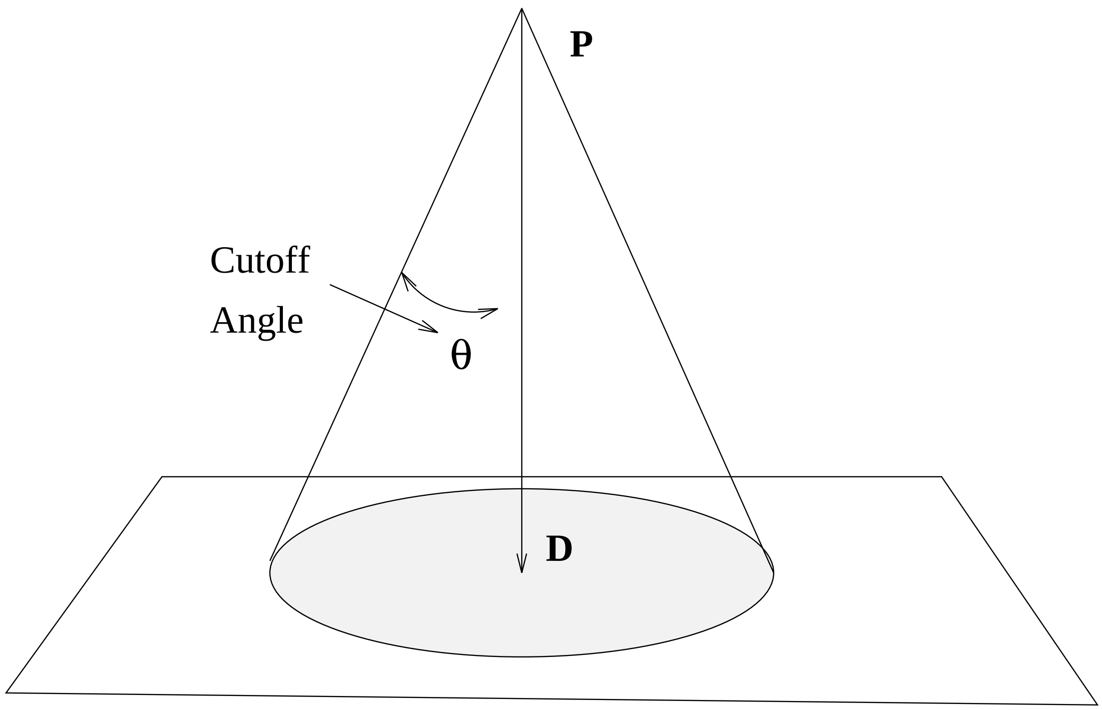


Figure 4.4: Spot Light

cylinder is no longer required as it will be the enclosure which will be lit and checked for intersection. For this to work, the sampling of diffuse light must check to a very fine level. An even better solution would place an area or spherical light source inside an enclosure. This would produce the smooth shadow required as well as lighting the interior of the enclosure.

4.3.5 Area Sources

Much of todays lighting consists of fluorescent strip lights contained inside some type of enclosure which scatter the light randomly. These can be modelled by light emitting polygons. This type of light source is not easy to sample quickly, because it could be thought of as a large number of point sources which produce umbra and penumbra, see figure 4.5. In the implementation, area sources of three or four sides have been included which can be of any shape except a concave quadrilateral. There are several ways in which the amount of light incident upon a surface due to this type of source can be calculated, of which all have been discussed in a different context in chapter 3. The methods include:

- Take a point in the centre of the source and use that ray as the shadow ray. This is completely inadequate, only treating the area as a point source.
- Take shadow rays chosen uniformly across the source and take the average intensity of all rays. This will produce smooth shadows, but can be subject to aliasing.
- Choose points on the source randomly. This will remove aliasing, but may give rise to clumping effects in parts of the light source. It is however very easy to implement.
- Jitter uniformly distributed points across the source, and again take the average intensity. This should remove any aliasing effects.
- Perform multidimensional random sampling. This will remove all aliasing and clumping effects, and will give very smooth shadows, but will be slower to run especially for irregularly shaped light sources as the hyperplane splitting becomes less trivial.
- If an object partially occludes the source, then adaptively sample this area intensively in order to determine an accurate measure of the intensity incident.

These methods take different amounts of time to compute and the above order is from fastest to slowest. Choosing which method to employ may depend upon

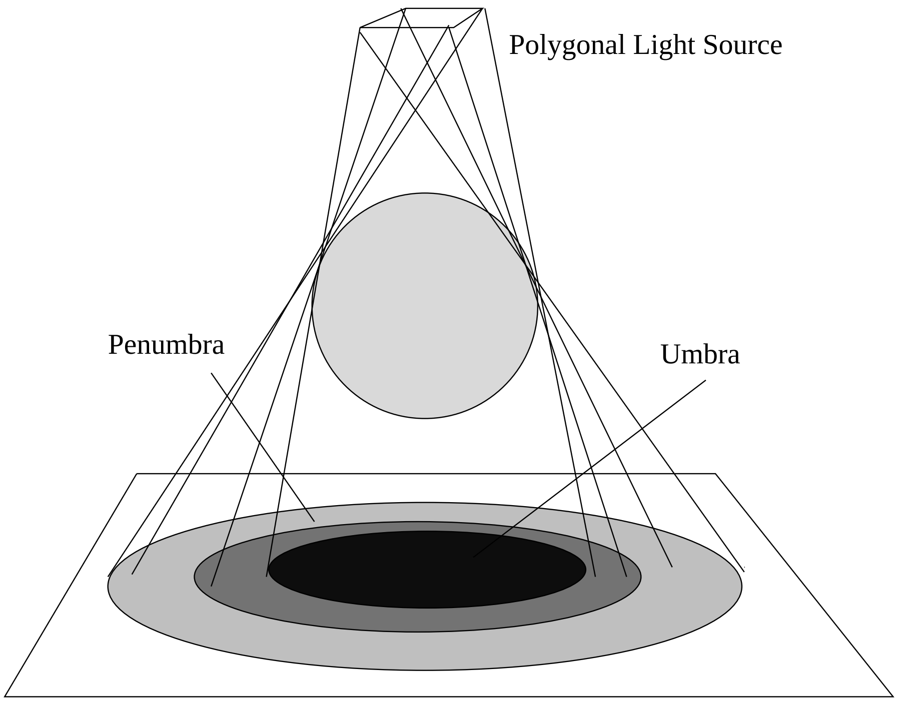


Figure 4.5: Area Source

a number of parameters, such as the distance of the source from the surface, the size and shape of the light, and the intensity. If the source is not very bright, or it is a long distance from the surface, then the variation in intensity across a shadows penumbra will not be very large. In that case then, not as many rays need be cast to reduce the variance. For sources with a small area, random sampling may be the best approach since the penumbra region will not be as wide as that of a larger source, and the number of shadow rays cast towards it may be based on the area. Multidimensional sampling is a good method as the source becomes well sampled, however for sources of irregular shape, calculating the hyperplane split and subsequent random point is computationally expensive. In the implementation, sources are sampled randomly with the number of shadow rays dependent on the area of the source.

4.3.6 Circular and Spherical Light Sources

Spherical sources, as with area sources, prove problematic to sample correctly. One way of proceeding is to model the sphere as a disc perpendicular to the shadow ray. This is discussed by Lange [32] where points are chosen randomly on the disc. One draw back to this is that during the shadow test between the source and the light, the sphere itself will always be intersected. This can be overcome by making the shadow tester ignore the sphere.

4.3.7 Indirect Lighting

In many new buildings the emphasis is towards illumination with a lot of indirect lighting to cut down glare, figure 4.6. This type of lighting is almost impossible for a conventional ray tracer to model. In order for path tracing to model scenes of this kind it is often the case that only adaptive sampling for the diffuse rays will give a good solution in a reasonable amount of time, this need is illustrated in the results section.

4.4 Acceleration Techniques

Probably the most important aspect of designing a useful tracer, is to render the image in a sensible time. This next section outlines all the non sampling techniques which may be used to speed up image creation. All of the techniques described have been used to implement the path tracer, except for the voxel technique which was omitted in favour of ordered bounding box checking.

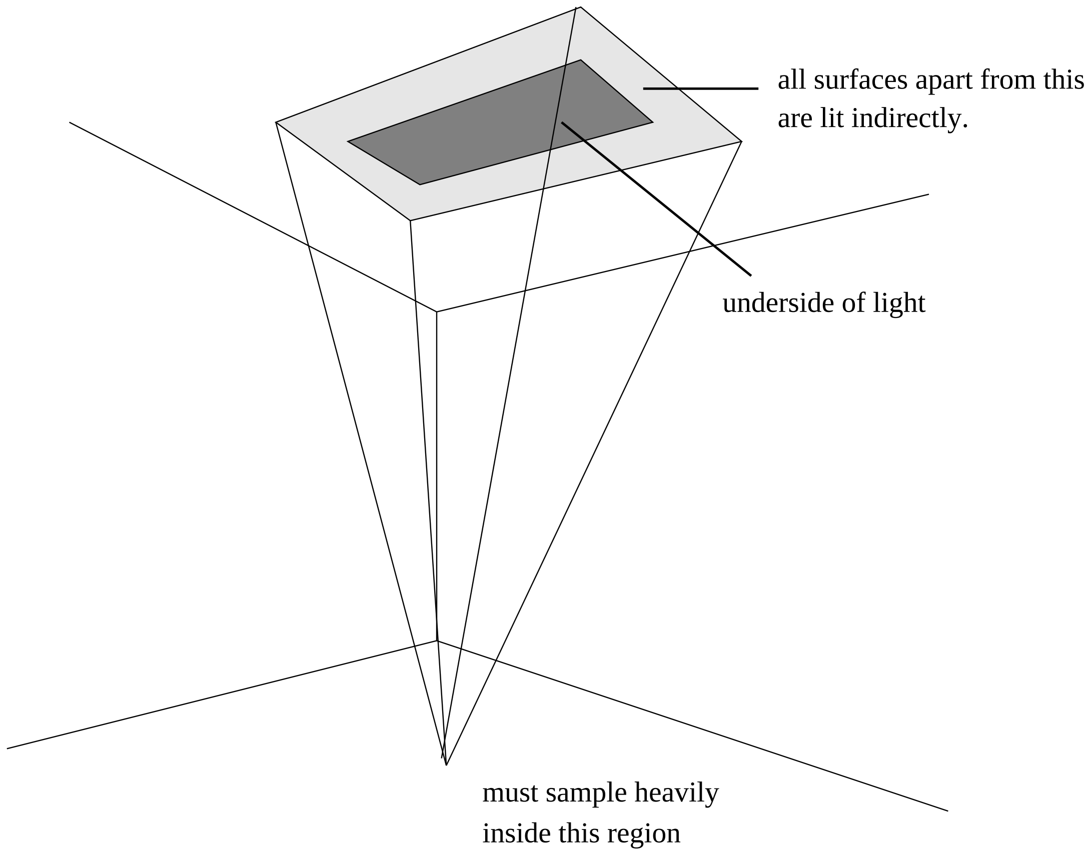


Figure 4.6: Sampling for Indirect Lighting

4.4.1 Bounding Boxes

For relatively simple objects such as spheres, planes, and simple polygons, the intersection algorithms are fast, however for complex shapes the execution time soon becomes slower. It was therefore suggested by Kajiya [28] that objects be placed inside an axis aligned bounding box, and this box tested first for intersection. If the ray intersects the box then the object contained is tested for intersection. The advantage of this is that intersecting an axis aligned box is very simple and fast. So for scenes containing many complex objects, the intersection time of a ray with that scene is decreased enormously. In his paper Kajiya continued to say that by extending the bounds to more than three dimensions, the bounding planes can be made to fit convex hulls arbitrarily tightly. Only the box

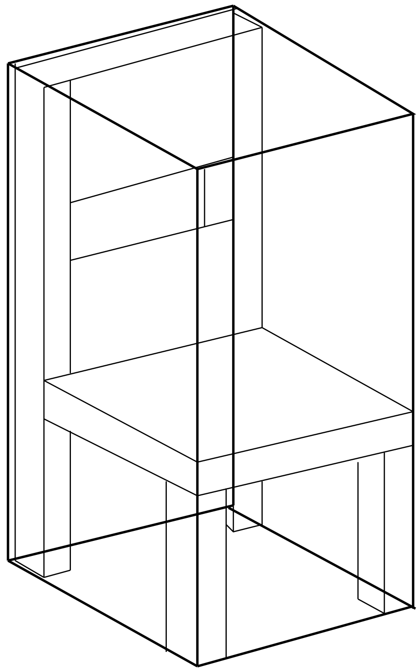


Figure 4.7: Bounding Box Surrounding a Hierarchical Object

method has been used so far but it would be an easy extension to make, the only problem being how to calculate a tight fitting enclosure for a given object.

4.4.2 Hierarchical Objects and Bounding Boxes

It may sometimes be the case that an object is made up other objects. For example a chair could be made of several transformed boxes as in figure 4.7. In cases like this it is sometimes easier to place a bounding box around the entire object and test this first, then if the ray does not intersect, then none of the objects contained within the box need be tested, thus speeding up the process. For complex objects a hierarchy structure of object bounding boxes may be set up, so that the initial bounding box may contain any number of bounding boxes, which in turn may contain many more. In this way for a very complex object, very few ray intersections need be carried out in comparison with the number of

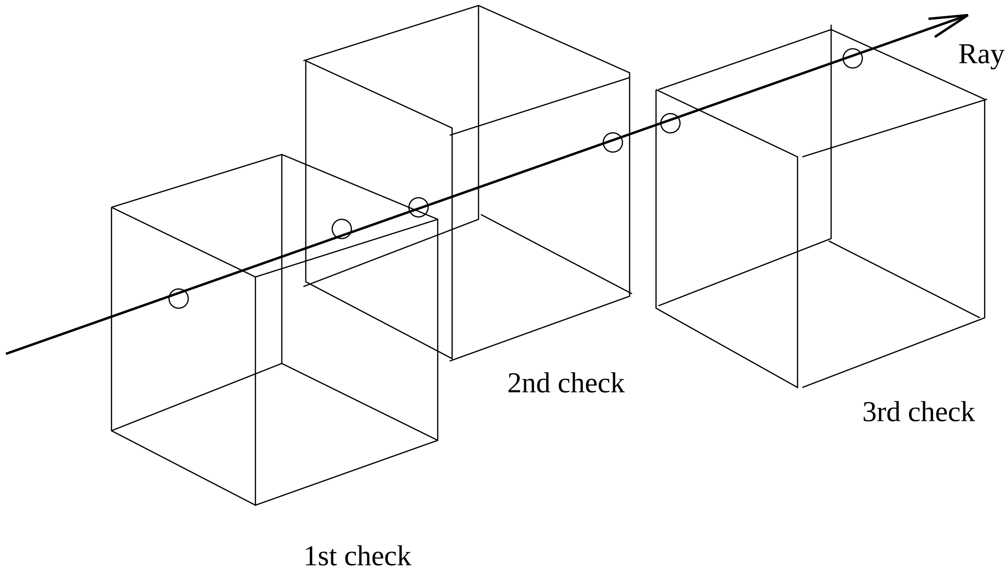
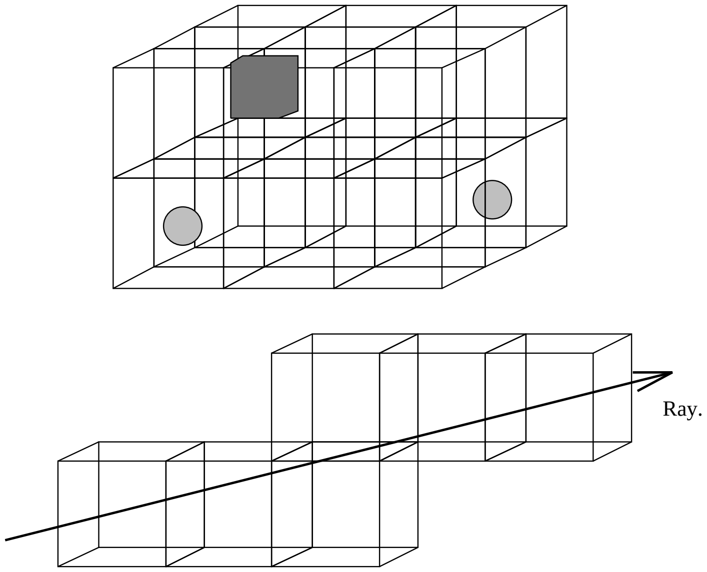


Figure 4.8: Ordered Checking of Bounding Boxes

objects contained within the initial box.

4.4.3 Traversal of Bounding Boxes

In a complex scene a ray may intersect with many bounding boxes. Normally each of these bounding boxes and their contents must all be checked before the closest object can be determined. Kajiya [28] in the same paper as above proposed that a record be kept of the intersection distances of the bounding boxes, and then check the contents of the closest box first, see figure 4.8. If one or more objects is found to be intersected in the first box, then no more bounding boxes need be checked, as any other objects will be behind those already found. The closest object inside the first box can be taken as the nearest intersected object. Difficulties arise if any bounding boxes intersect, or if one box lies entirely within another, but this can be overcome by arranging the data structure in such a way as this can be checked.



Voxels are checked in order in which ray passes through.

Figure 4.9: Voxels

4.4.4 Voxels

This technique involves a large amount of preprocessing to reduce the bounding box checking time. The entire scene is split into a number of aligned boxes, or *voxels*. The contents of each box are recorded and stored. Some objects may lie inside more than one box, and many boxes may be empty. When a ray is cast through the scene, all the voxels which are intersected are checked in order similar to the ordered bounding box checking. Thus inside each voxel all objects are intersected and the minimum distance found inside that voxel. The advantage over the ordered bounding box checking depends on the type of scene that is being traced. If in preprocessing a calculation is made to find the order in which the voxels will be intersected for any given ray, then the method is much improved. One disadvantage is that any one voxel may contain a large number of objects, again depending on the type of scene being considered.

4.5 Texturing

It is very often the case that in order to create a realistic image, an object having just one colour is inadequate. Texture mapping involves taking an image and mapping it onto a surface (such as putting a picture on a wall), while procedural textures are algorithms which mimic material properties such as wood or granite.

4.5.1 Texture Mapping

In a scene it is often necessary, in order to create realism, to include objects with extra surface detail than a description of the general properties. This could be for many reasons, such as putting a number on a billiard ball, or modelling scratches or roughness which are not easily modelled algorithmically. The basic idea is to take an image in a form such as TIFF or GIF, and map the colour points of this onto a surface. Then, the value of the (r,g,b) colour at that point is scaled to be within the range (0,1) and used as the diffuse reflection coefficient at that point. A smaller value is also given to the specular reflection coefficient which is 0.3 multiplied by the diffuse value. The easiest objects to texture map are spheres, and 3 or 4 sided polygons, which have been included in the implementation.

Mapping onto a Sphere

Given an intersection point on a sphere, we need to calculate a u,v position on an image. The technique used for this is called spherical inverse mapping and is shown in figure 4.10. This is described by Haines [22] as follows.

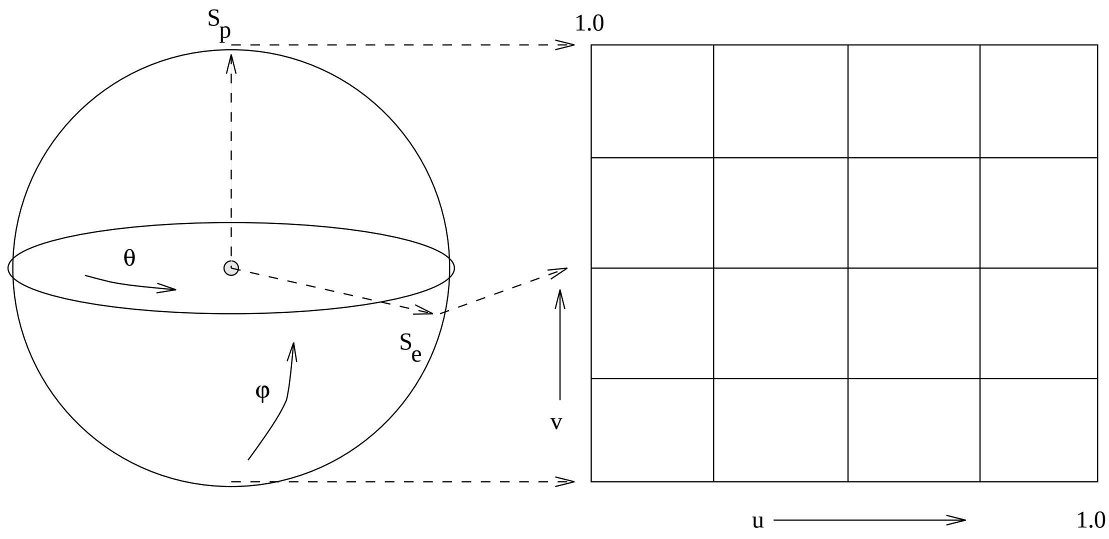


Figure 4.10: Spherical Inverse Mapping

The parameters which are required are the sphere unit normal $\hat{\mathbf{S_n}}$, at intersection point $\mathbf{R_i}$, and a description of the sphere and its axes such that $\hat{\mathbf{S_p}}$ is a unit vector from the spheres centre to the north pole, and $\hat{\mathbf{S_e}}$ is a unit vector from the spheres centre to a reference point on the equator. So $\hat{\mathbf{S_p}} \cdot \hat{\mathbf{S_e}}$ is equal to zero.

First find a latitudinal parameter

$$\phi = \arccos(-\mathbf{S_n} \cdot \mathbf{S_p})$$

and then the v position on the texmap is given by $v = \phi/\pi$.

If v is either equal to one or zero then the point is at the top or bottom of the sphere, and we can set u to be equal to 0 for consistency. Otherwise continue to calculate u for which we need a longitudinal parameter:

$$\theta = \frac{\arccos(\hat{\mathbf{S_e}} \cdot \hat{\mathbf{S_n}})}{\sin(\phi) * 2 * \pi}$$

Now take the cross product of the two sphere axes defining angles, and compare this direction with the direction of the normal such that:

- if $((\hat{\mathbf{S_p}} \wedge \hat{\mathbf{S_e}}) \cdot \hat{\mathbf{S_n}}) > 0$ then $u = \theta$,

- else $u = 1 - \theta$.

The cross product is used to determine which side of the $\hat{\mathbf{S_e}}$ vector the intersection point $\mathbf{P}$ lies upon, and could be calculated in advance.

Mapping onto a Polygon

This is called *convex quadrilateral inverse mapping*. Once an intersection point has been found inside a polygon, the associated colour from the colour map must be found, which is specified by a (u, v) co-ordinate, where both u and v are a number between 0 and 1. The method for this is quite involved. Haines [22] describes one way in which the solution can be found, but another way, devised by the author has been used. Haines' method solves the problem using all 3 coordinates in his calculations, however the method described below maps the points onto a 2-dimensional plane before doing the calculations. This means that all calculations may be done with only 2 coordinates.

Basically what I attempt to calculate are the vectors at the intersection point as shown by 4.11, which can then be used to find the u and v coordinates required.

Unit vectors are calculated along each of the polygon sides. Then before any further steps are taken, the opposite sides are checked to see if they are parallel, as this speeds up calculation considerably. If they are near parallel then the vector at the intersection point is equal to one of them, and the next step may be omitted.

Take two of the opposite sides and calculate the line equation of each by simple

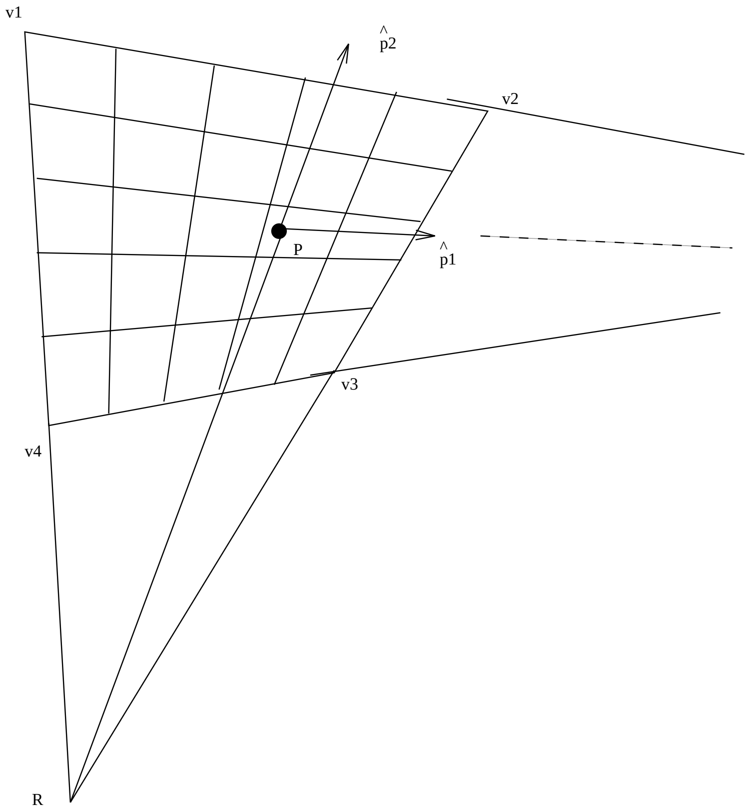


Figure 4.11: Inverse Quadrilateral Mapping

geometry, then the intersection point **R** of these lines can be found. Once this is known a vector from **R** to the ray intersection point can be calculated and normalized. This should also be done for the other set of opposite sides.

Now considering the geometry in figure 4.12, the positions on the colour map may be found. The v parameter is given by following:

$$\theta = \arccos(\hat{\mathbf{pnvec}}.\hat{\mathbf{p1}})$$

$$\phi = \arccos(\hat{\mathbf{v4}}.\hat{\mathbf{p1}})$$

where $\hat{\mathbf{pnvec}}$ is a the normal of **pvec**, then with the use of the sin law:

$$v = \frac{(|\mathbf{pvec}| \sin\theta)}{\sin\phi|\hat{\mathbf{v4}}|}$$

The u parameter may be found similarly.

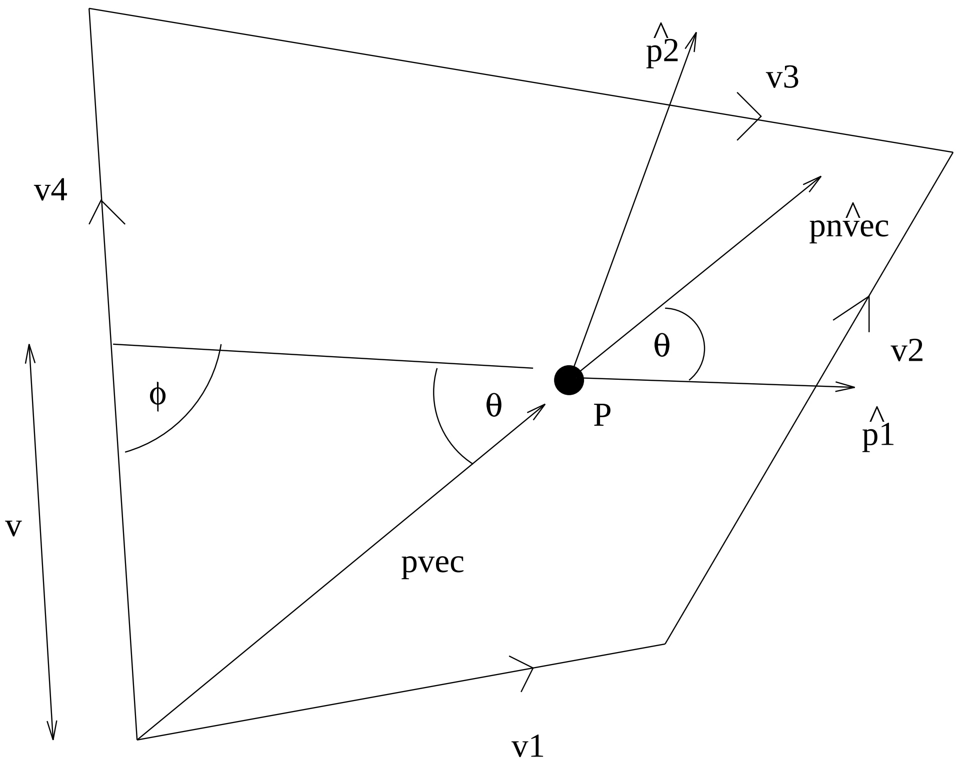


Figure 4.12: Mapping Geometry

4.5.2 Procedural Textures

Although texture mapping allows any image to be placed onto an object, if it is desirable for the object to appear for example wooden, then the orientation and choice of texture map becomes difficult. Procedural textures get around this problem by using details such as the intersection points normal, position, and incident ray to calculate colour points which will give the object the appearance of being carved out of a particular material.

One of the requirements of creating a real texture is to make it appear as though it is not uniform across the whole object, which means that some sort of randomness must be introduced. Perlin [38] discusses the way in which a noise function may be written. This function has been implemented by modifying code written by Setzer [43], which creates a pink noise function. Using this, algorithms have been constructed to model textures such as wood, granite, and marble by modifying the algorithms given in the Renderman Companion [48].

4.5.3 Environment Mapping

A mention of this technique will be made as it is quite easy to implement [26]. It is a way of texture mapping reflections and refractions. Given an object, which may be made up of many other objects and is for example at the centre of a room, then especially for animation, exact reflection is unnecessary. This is because for very complex curved objects, it is less important to see what is actually reflected in the object, but rather to have reflections that give a better understanding of the curvature of the object. The major advantage of environment texture maps is that they provide some realism at much less cost than a full ray tracing solution. For this reason however they should not really have a place in a realistic implementation.

A virtual object (usually something simple such as a sphere) which has some texture map associated with it may be placed around an object. When an intersection is made with the real object, the colour of the reflected ray is found with reference to the virtual object, and no consideration of the other objects in the scene is taken. Thus a virtual sphere mapped with an environment such as land and sky could be projected onto a car to give it the feel of the environment that surrounds it.

Chapter 5

Results and Conclusions

This chapter includes photographs which have been taken from the screen of a Hewlett Packard 24-bit workstation which illustrate some of the effects that can be achieved using the techniques already described. Conclusions and suggestions for future work are also included.

5.1 Results

The first few plates are organised such that they illustrate the different aspects of light interaction which can be achieved using path tracing, and then continues to bring these together into more complex images. Time restrictions have meant that scenes with very complex environments have not been included, and also a full testing of the variance techniques used has not yet been possible. For this reason the maximum number of rays per pixel, and thus the execution time is often very high.

Plate Number	Max No. Rays/Pixel	Maximum Ray Depth	No. of Primitives	Image Size	Shadow Rays(m^2)	CPU Time(hrs)
1	50	3	7	400x400	4	6.0
2	32	3	3	200x200	4	1.1
3	8	3	12	200x200	1	0.75
4	32	3	12	200x200	1	1.5
5	16	4	4	200x200	4	3.0
6	50	3	8	400x400	4	6.0
7	50	4	8	400x400	4	8.0
8	50	3	10	50x50	1	0.15
9	100	3	10	50x50	1	0.3
10	200	3	10	50x50	1	0.6
11	50	3	16	400x400	4	10.0
12	n/a	3	16	400x400	4	n/a
13	16	3	141	400x400	4	2.0
14	48	3	141	400x400	4	4.0
15	20	1	141	400x400	4	0.3
16	50	3	141	400x400	4	7.0
17	48	3	255	400x400	4	8.0
18	50	3	255	400x400	4	10.0
19	50	3	620	400x400	4	11.0
20	64	3	3000	400x400	4	48.0

Table 5.1: Plate Statistics

5.1.1 Plates

Table 5.1 shows various statistics for the scenes which have been modelled, indicating the maximum number of rays per pixel, maximum ray depth, number of object primitives, image size, no of shadow rays per square metre, and time taken to run on a Hewlett Packard 9000/735 series workstation. It should be noted that in taking photographs of the screen, some image quality has been lost. This has resulted in colour bleeding being less apparent for some brightly lit surfaces.

emptybox

Plate 1: Empty Room Showing Colour Bleeding

Diffuse Reflections

The first 4 plates show only diffuse surfaces in order to illustrate diffuse inter-object reflection, and thus colour bleeding. Plate 1 shows the inside of an empty room, which is light grey (0.8, 0.8, 0.8) in colour, except for the red (0.8, 0.1, 0.075), and blue (0.075, 0.1, 0.35) walls opposite each other. A rectangular white light source is situated on the ceiling. It can be seen that colour bleeding occurs at distances near to the walls. This is especially noticeable on the ceiling, which is being lit indirectly.

Plate 2 has a sphere resting on a plane with an area light source casting a shadow onto the plane behind it. Penumbra regions on the shadow are clearly visible, and the rear of the sphere has been indirectly lit.

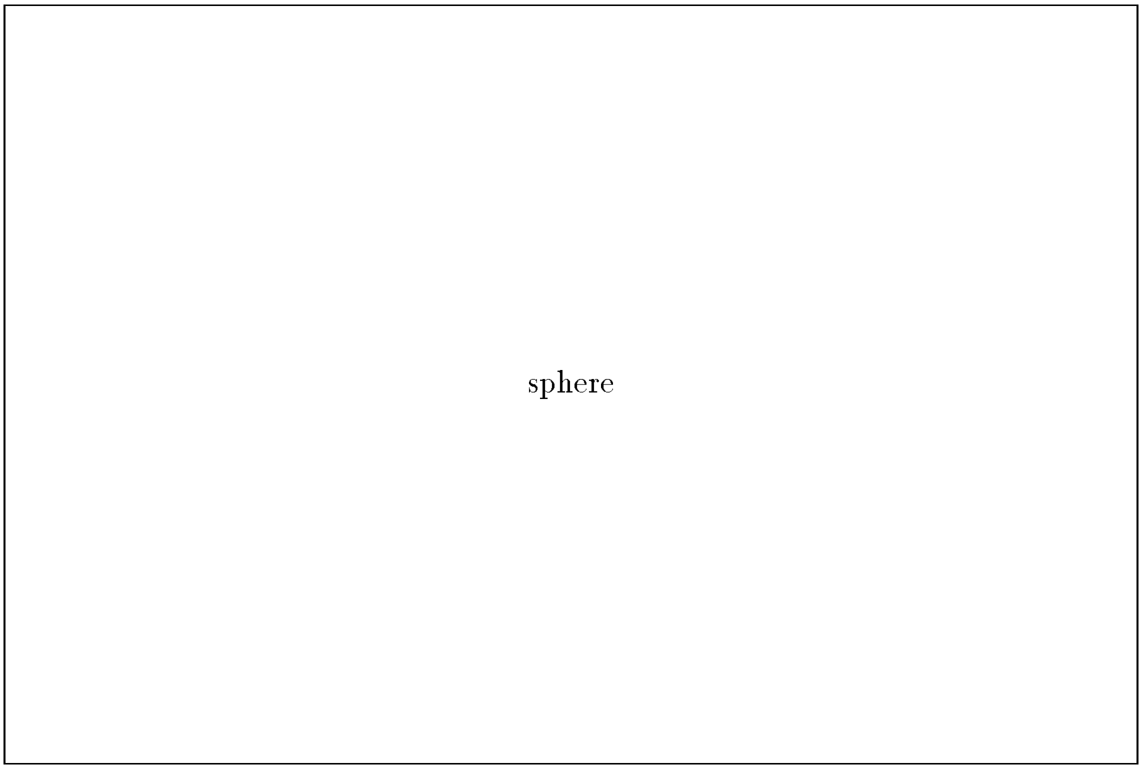


Plate 2: Sphere on a Plane

Plates 3 and 4 show the same scene with a grey cube placed in the centre of the room, the floor colour has also been changed to green. The difference between the two images is the number of rays per pixel that have been cast, as can be seen from table 5.1. Even with a low number of rays the surfaces have picked up a reasonable amount of colour. For plate 4 where many rays have been cast the variance across the image is negligible. It is the case however that this is a relatively simple scene, where little variance occurs, and some of the later images require further sampling.

cubox1

Plate 3: Low Number of Rays

cubox3

Plate 4: Increased Number of rays

juggreflection

Plate 5: Hazy Reflections

Specular Effects

Plates 5 and 6 illustrate specular reflection. Plate 5 shows a mirrored half cube, with a texture mapped sphere in front. The separate sides have different Phong [39] exponents and so exhibit varying degrees of haziness. There are two point light sources present, one at the viewing point, and one in the back corner behind the sphere. The mirror with the lowest Phong exponent reflects a lot of the light in an almost diffuse manner, and the reflection is very blurred. Plate 6 has a mirrored sphere inside the red and blue walled room. Note how colour bleeding is still apparent in the reflections.

Plate 7 now shows a transmitting sphere inside the red and blue walled room. The patch of light on the floor should be noted, as this is caused by diffuse rays mirrorsph

Plate 6: A Mirrored Sphere

being followed from the floor, passing through the sphere, and intersecting with the area light source. It should also be noted that the sphere inverts the light passing through it, giving rise to the refracted images of the red and blue walls appearing on opposite sides. The diffuse reflections of the refracted rays are not as smooth as for the first generation intersections because all higher generation rays only follow a single ray, and do not attempt to sample the diffuse distribution correctly.

Plates 8 to 10 consist of 3 test images, scaled up 8 times, each with a different number of rays cast. All light in the scene is indirect as the viewing direction points away from any direct illumination, see figure 5.1. It can be seen from the plates that the number of rays had to be very large before the variance was reduced to any kind of acceptable level. This test case then shows a need to caustics

Plate 7: A Transparent Sphere

consider adaptive sampling, as the amount of work to produce a good image of this type becomes excessive.

Plate 11 shows another test scene to illustrate colour bleeding. The box on the left is yellow(0.9, 0.8, 0.1), while the right hand box is grey. The colour bleeding effects are very noticeable in this image. This is because the sides of the boxes have no direct light incident upon them. This scene has also been used to illustrate the effect of controlling the number of rays cast for each pixel. Plate 12 shows the same image but with the number of rays cast used as the pixel colour. The brightest areas of the picture thus correspond to the most heavily sampled regions. It can be seen that the regions most heavily sampled occurred along object edges, and in penumbra regions. Regions of high indirect lighting were sampled less heavily, while uniform sections were sampled the least.

lroom1

Plate 8: 50 Rays per Pixel

lroom2

Plate 9: 100 Rays per Pixel

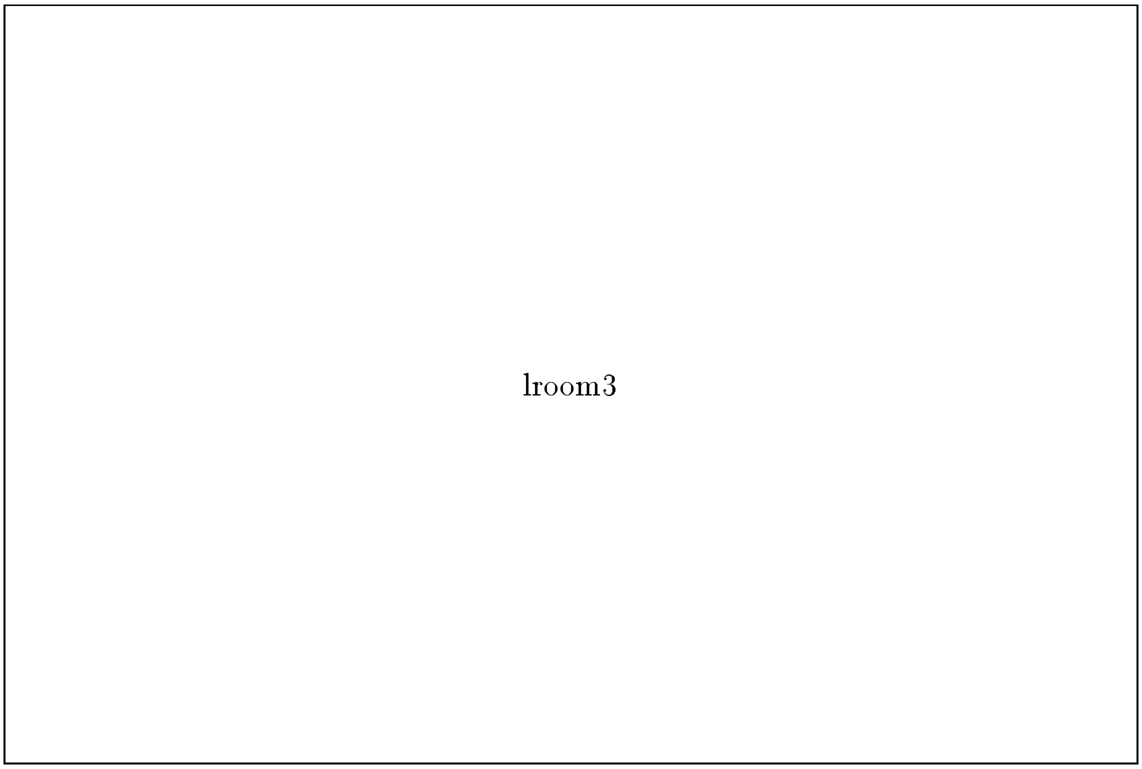


Plate 10: 200 Rays per Pixel

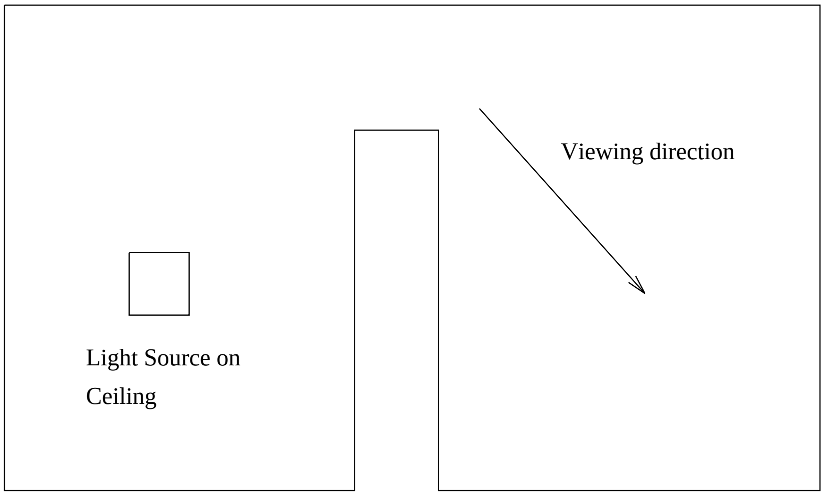


Figure 5.1: Plan of Indirect Lighting Test Case

standard boxes

Plate 11: Standard Box Scene

greyscale

Plate 12: Greyscale Showing the Relative Number of Rays per Pixel

tablechairrough

Plate 13: Rendered With Few Rays per Pixel

Plates 13 and 14 both show the same scene which has a polygonal light source above the table, rendered with a different number of maximum rays. It can be seen how the second image has very little variance across the penumbra regions and that the table and chair legs have been indirectly it to a greater extent. Plate 15 has the same scene again rendered with two procedural textures, one of granite, and the other a with a checkerboard routine. The run time is fast for this image as the rays were only followed to a depth of one.

Plates 16 and 17 look at a room which has two polygonal light sources present. The attenuation of light with distance is not present within this scene, and for this reason the walls do not appear darker towards the rear of the room.

In plates 18 to 20 the kind of effects that can be achieved for realistic scenes is tablechairsmooth

Plate 14: Increased Number of Rays

shown. Time restrictions have not allowed very good modelling of scenes as yet but a good idea of the realism possible can be appreciated.

5.2 Conclusions

An investigation and implementation of a Monte Carlo path tracer is presented. Specific methods are chosen for the sampling, illumination, lighting, and speed up techniques, from those considered, and used within the implementation.

An illumination model is used to model diffuse, reflective, and transmissive surfaces with a number of different lighting techniques, including area sources. These have been used to solve the rendering equation [29], and produce photorealistic texturetable

Plate 15: Procedural Textures

emptysplit

Plate 16: An Empty Room

railroom

Plate 17: Room with Railings

fullroom

Plate 18: A Room with Furniture

pictureroom

Plate 19: Room From a Different Angle

plant

Plate 20: A Complex Polygonal Scene

images.

Various object primitives are used to form complex objects which can be intersected quickly using hierarchical data structures and bounding boxes. These objects are used to design scenes which exhibit realistic lighting effects such as smooth shadows, diffuse inter-object reflections, and thus colour bleeding, as well as textured surfaces, hazy reflections, and caustics.

Monte Carlo methods have been used to adaptively control the number of rays per pixel, determine which type of path to follow, control the number of specular rays, and determine which direction to shoot a next generation ray. A discussion of the feasibility of adaptively sampling the diffuse directions has also been presented, however because of time restrictions a program to implement this is still in development.

The plates presented in chapter 5 illustrate that the methods used in the implementation certainly work, although time has not yet allowed a full testing of these.

5.3 Future Work

Ray tracing techniques have been very heavily researched, and further investigation into these could increase the speed of the path tracer as this is one of its main drawbacks, however a greater source of speed and accuracy might be to investigate further the Monte Carlo methods used. The main areas that this would be applied to are:

- *Pixel Sampling*: The number of initial rays cast through the pixel could be reduced if the cost to the image quality was not significant. This would decrease the run times greatly.
- *Shadow Testing*: If an improved way of controlling the number of shadow rays cast could be developed then further increases in speed would be made.
- *Adaptive Sampling of the Diffuse Directions*: The number of rays cast diffusely from an intersection point is a large source of computation, and also image noise. If more research were carried out in this area then not only would time benefits be gained, but image quality would be improved.

Path tracing has not enjoyed the wide success of other global illumination models, most notably radiosity, however the images produced are certainly of a realistic nature, and thus perhaps when hardware improvements are made in the future it may be considered a useful tool for image creation.

Appendix A

A Summary of Intersection Methods

This appendix outlines the main object intersection routines used in the implementation, of which the sphere, polygon, and ellipsoid intersection can be found in Glassner [19], the others have been devised by the author, based to a large extent on these. Throughout this, the notation used to represent rays and points will be of the form.

- Ray origin $\mathbf{R_o} \equiv (X_o, Y_o, Zo)$
- Ray direction $\hat{\mathbf{R_d}} \equiv (X_d, Y_d, Zd)$
- A ray is the set of points in front of the ray origin, on a line represented by: $\mathbf{R}(t) = \mathbf{R_o} + \hat{\mathbf{R_d}} * t$, where $t > 0$ and is the intersection distance.
- Intersection point $\mathbf{P_i} \equiv (x_i, y_i, z_i)$ with normal $\hat{\mathbf{N_i}}$ to surface.

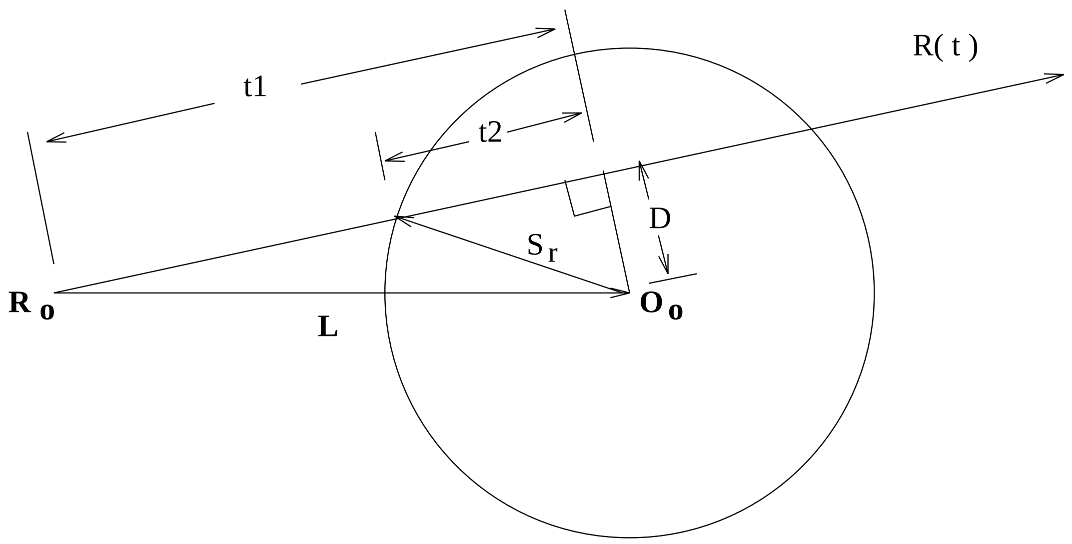


Figure A.1: Sphere Geometry

A.1 Intersection of a Sphere

There are two methods available for intersecting a sphere, an algebraic and a geometric methods. The geometric method is more efficient than the algebraic for a few reasons. Firstly it determines whether or not the ray intersects the sphere before actually working out the intersection point. For example, the first comparison test determines whether or not the ray points towards the sphere, or away from it, which is a good idea as a random ray will only point towards a sphere for half of the time. Let the sphere origin be $\mathbf{S_o} \equiv (X_s, Y_s, Z_s)$.

The steps to take for the geometric method are as follows and reference should be made to figure A.1.

- Determine whether the rays origin is outside the sphere by calculating the distance squared from the ray origin to the centre of the sphere L^2. The vector to the centre of the sphere from the ray origin is $\mathbf{L} = \mathbf{R_o} - \mathbf{S_o}$ which gives $L^2 = |\mathbf{L} \cdot \mathbf{L}|$.

- If L^2 is less than the spheres radius squared, then the origin lies inside the sphere and calculation can be stopped, unless the sphere is transmissive. Otherwise calculate the rays closest approach to the sphere.
- If the distance $t1 = \mathbf{L} \cdot \hat{\mathbf{R_d}}$ is less than zero then the ray origin lies beyond the sphere centre, and calculation may be stopped, else continue.
- The distance $t2^2 = S_r^2 - L^2 + t1^2$ is the half chord distance squared, and if this is greater than zero then the ray must hit the sphere with an intersection distance of $t = t1 - \sqrt{t2}$

Given the intersection point $\mathbf{P_i}$ the unit normal at that point is given by:

$$\hat{\mathbf{N_i}} = \frac{\mathbf{P_i} - \mathbf{S_o}}{|\mathbf{P_i} - \mathbf{S_o}|}$$

A.2 Intersection of a Polygon and Plane

When intersecting a polygon, a check must first be done to see if the ray intersects the plane in which the polygon lies. When this has been done and the intersection distance found, then a process of inside outside testing can be done to determine whether or not the intersection point lies inside the polygon.

A.2.1 Intersection of a Plane

The equation of a plane can be given by:

$$Ax + By + Cz + D = 0$$

where $A^2 + B^2 + C^2 = 1$, and D is the distance from the coordinate system origin to the plane whose sign determines which side of the plane the origin is. The normal to the plane is $\mathbf{P_n} = (\hat{\mathbf{A}}, \mathbf{B}, \mathbf{C})$.

The ray equation may be substituted into the plane equation and the intersection point, if one exists, calculated. This would give the intersection distance t as:

$$t = \frac{-(\mathbf{P_n} \cdot \mathbf{R_o} + \mathbf{D})}{\mathbf{P_n} \cdot \mathbf{R_d}}$$

If this is split up into two stages however, a check can be performed. If $|\mathbf{P_n} \cdot \mathbf{R_d}| < 0 + k$, where k defines a small number defining the tolerance required, then the ray is parallel to the polygon and no intersection occurs. Otherwise the ray intersects the polygon at t, but if $t < 0$ the point is behind the ray origin.

If the polygons are back facing (meaning that they have a colour on both sides) then it must be made sure that the normal is pointing in the direction in which the light calculations will give a colour. If $\mathbf{R_d} \cdot \mathbf{N_i} > 0$ then the normal must be negated.

A.2.2 Polygon Intersection

In order to find out whether the point on the plane is inside the polygon, an *inside outside test* must be performed. The idea is to project the intersection point, and all of the polygon points, onto a two dimensional plane. The plane chosen to map the points onto is one of the axis planes. Instead of actually mapping the co-ordinates onto a plane, one of the x, y, or z components may be discarded. The co-ordinate which is thrown away is the component of the plane normal which is of the greatest magnitude.

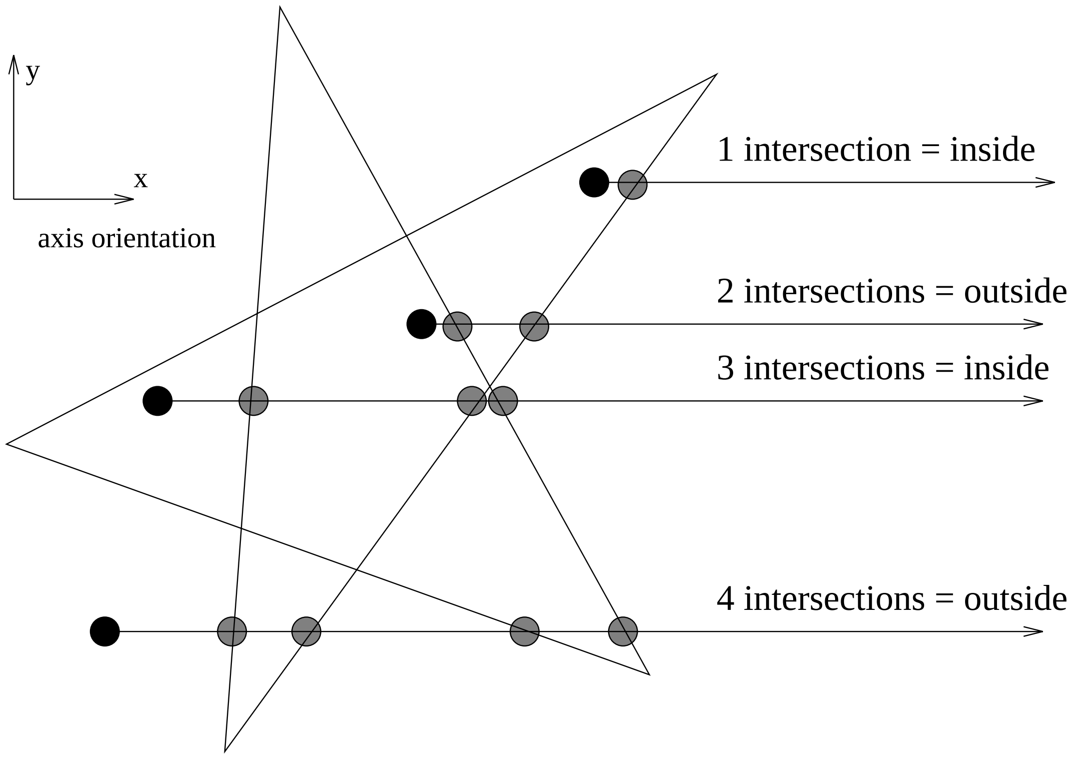


Figure A.2: Inside Outside Testing

Now that the polygon has been projected onto the plane the inside-outside test is quite simple. Firstly, all the points should be translated so that the intersection point is at the origin. It is then imagined that a ray starts from the origin, and proceeds along one of the projected plane axes. Each edge of the polygon is tested against the ray, and any edges crossed are noted, this can be seen in figure A.2. Two checks may be carried out to see whether the ray crosses an edge. Assuming that the ray follows the x axis, then if the x component of both ends of an edge are positive, then the ray must cross the edge. If only one is positive then the ray may cross the edge. In this case compute the intersection of the edge with the axis. If it is positive then the edge must cross the axis. If the total count of crossings is odd, then the point is inside the polygon, the normal of which is already known.

A.3 Intersection of an Axis Aligned Box

This intersection routine forms the heart of all of the intersection tests as nearly all objects have a bounding extent.

Since the box is axis aligned, it is made up of three sets of parallel planes, or *slabs.* If the ray passes through all three slabs then it must intersect the box. The box is defined as two points, a maximum and a minimum extent. Let the box have a minimum extent of $\mathbf{B_l} \equiv [X_l, Y_l, Z_l]$ and a maximum extent of $\mathbf{B_h} \equiv [X_h, Y_h, Z_h]$, and also set two variables $t_{near} = -\infty$, and $t_{far} = \infty$, which will contain the near and far intersection distances of the box.

Now for each of the set of planes associated with X,Y and Z (shown here for the $\alpha = X, Y, andZ$ plane) do the following:

- If the α direction of the ray α_d is equal to zero then the ray is parallel to the planes so, if the origin α_o is not between the slabs then no intersect.
- Else, if the ray is not parallel to the planes, then:

  Calculate the intersection distances of planes:

  $$t_1 = (alpha_l - alphao)/alpha_d$$

  $$t_2 = (alpha_h - alphao)/alpha_d$$

- If $t_1 > t_2$ swap t_1 and t_2.
- If $t_1 > t_{near}$, set $t_{near} = t_1$.
- If $t_2 < t_{far}$, set $t_{far} = t_2$.
- If $t_{near} > t_{far}$, box is missed so not intersect.

- If $t_{far} < 0$, box is behind ray so no intersect, end of loop.

If the box is hit the intersection distance will be equal to t_{near}.

A.4 Intersection of a disc

Intersection of a disc is very simple. Initially intersect the ray with the plane in which the disc lies. If the ray intersects the plane then calculate the distance from the intersection point of the plane to the centre of the disc. If that distance is less than the radius of the disc, then the ray must intersect. The normal is already known.

A.5 Intersection of a Quadric

Quadrics are quite complicated to intersect and take a large amount of computing time in comparison to other methods. To do this, the ray equation must be substituted into the equation for a quadric, which is described in Glassner [19].

The intersection of a cylinder involves intersection with the ends of the cylinder as for the disc and using a much simplified quadric intersection routine for the main body of the cylinder.

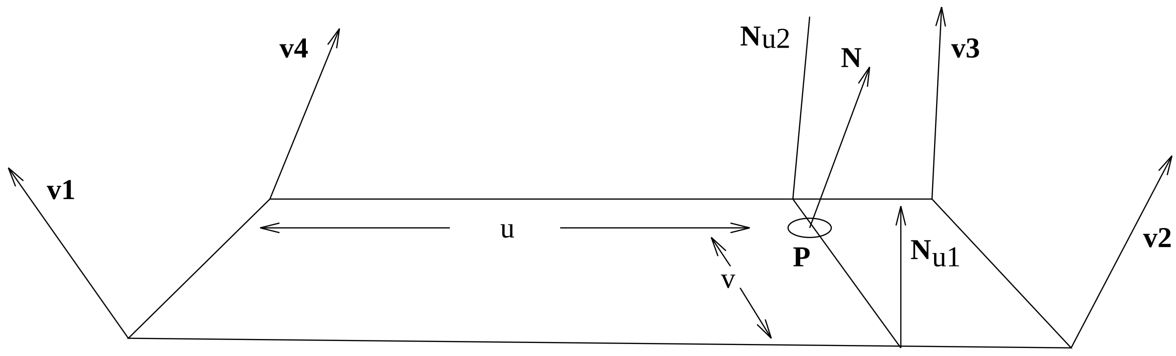


Figure A.3: Interpolation of Vertex Normals

A.6 Polyhedra

A polyhedron is an object consisting of more than one polygon. It is defined by a set of points followed by a list of vertexes defining each of the polygons. A point may have more than one polygon associated with it. A polyhedron may be entirely enclosed by a bounding box to speed up its intersection. One other way of speeding up intersection for especially complex polyhedra is to split the object into a number of separate polyhedra, whose bounding boxes may each be tested for intersection, thus needing many less polygon intersections.

Polygons can be used to model very complex shapes although the appearance of that shape will not look smooth. One way of producing realistically smooth complex objects is to interpolate the vertex normals of the polyhedron over the polygon in question so that it appears contoured.

Given a 3 or 4 sided polygon with vertex normals, one way in which to calculate the interpolated normal, illustrated in figure A.3, is described by Phong [39]. Note that although the figure appears to show a rectangular polygon, quadrilaterals of any shape are allowed. If the polygon has three sides then one vertex may be repeated, or a separate module designed especially for that purpose.

Two numbers u and v are calculated, such that each is a percentage of the distance of the intersection point along opposite sides of the polygon. This is exactly the same as for texture mapping to specify the position of a point within a polygon. The idea is to interpolate two sets of adjacent vertex normals, and then interpolate between the new ones, to find the final normal $\mathbf{N}$. With reference to figure A.3:

$$\hat{\mathbf{N_{u1}}} = u\hat{\mathbf{v2}} + (1-u)\hat{\mathbf{v1}}$$

$$\hat{\mathbf{N_{u2}}} = u\hat{\mathbf{v3}} + (1-u)\hat{\mathbf{v4}}$$

$$\hat{\mathbf{N}} = v\hat{\mathbf{N_{u2}}} + \hat{\mathbf{N_{u1}}}(1-v)$$

and so $\hat{\mathbf{N}}$ can be used as the surface normal.

References

[1] M Abramowitz and I A Stegun. *Handbook of Mathematical Functions*. Dover Publications, 1972.

[2] J Arvo. Backward ray tracing. *Siggraph Developments in Ray Tracing seminar notes*, 1986.

[3] K E Atkinson. *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind*. Society for Industrial and Applied Mathematics, 1976.

[4] A H Barr. Global and local deformations of solid primitives. *Computer Graphics*, 18(3):21–29, 1984.

[5] J L Bentley and J H Friedman. Data structures for range searching. *ACM Computer Surveys*, 11(4):397–409, 1979.

[6] J F Blinn. Models of light reflection for computer synthesized pictures. *Computer Graphics*, 11(2):192–198, 1977.

[7] W J Bouknight. A procedure for generation of three-dimensional half toned computer graphics presentations. *Communications of the ACM*, 13(9):527–536, 1970.

[8] R L Cook. Stochastic sampling in computer graphics. *Computer Graphics*, 18(3):137–145, 1984.

[9] R L Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, 1986.

[10] R L Cook, T Porter, and L Carpenter. Distributed ray tracing. *Computer Graphics*, 18(3):137–145, 1984.

[11] R L Cook and K E Torrance. A reflectance model for computer graphics. *Computer Graphics*, 15(3):307–316, 1981.

[12] W Cowan. An inexpensive scheme for calibration of a colour monitor in terms of CIE standard coordinates. *Computer Graphics*, pages 315–321, 1983.

[13] M A Z Dippe. Antialiasing through stochastic sampling. *Computer Graphics*, 19(3):69–77, 1985.

[14] S Yakowitz et al. Weighted Monte Carlo integration. *SIAM Journal of Numerical Analysis*, 15(6):1289–1300, 1978.

[15] T A Foley, A van Dam, S Feiner, and J Hughes. *Principles and Practice.* Addison Wesley, 1990.

[16] N Gatenby. Radiosity in computer graphics, A proposed alternative to the hemi-cube algorithm. Master's thesis, University of Manchester, 1991.

[17] N Gatenby. *A Monte Carlo Radiosity Method Utilising Binary Space Partitioning Trees.* PhD thesis, University of Manchester, 1994. to appear.

[18] N Gatenby, M Preston, W T Hewitt, and A Knowles. The manchester scene description language (msdl) version 1.0. Technical report, Manchester Computing Centre, Computer Graphics Unit, 1992.

[19] A S Glassner, editor. *An Introduction to Ray Tracing*. Academic Press, 1989.

[20] A S Glassner, editor. *Graphics Gems*. Academic Press, 1990.

[21] C M Goral, K E Torrance, D P Greenberg, and B Battaile. Modelling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, 1984.

[22] E Haines. *An Introduction to Ray Tracing*, pages 35–77. Academic Press, 1989.

[23] R A Hall and D Greenberg. A testbed for realistic image synthesis. *ACM Transactions on Graphics*, 3(8):10–20, 1983.

[24] P S Heckbert. *Simulating Global Illumination Using Adaptive Meshing*. PhD thesis, University of California, 1991.

[25] P S Heckbert and P Hanrahan. Beam tracing polygonal objects. *Computer Graphics*, 18(3):119–127, 1984.

[26] Hewlett Packard. *ARTCore Programmers's Manual*, 1992.

[27] R V Hogg and A T Craig. *Introduction to Mathematical Statistics*. Macmillan Publishing Co., 4 edition, 1978.

[28] J T Kajiya. Ray tracing complex scenes. *Computer Graphics*, 20(4):269–277, 1986.

[29] J T Kajiya. The rendering equation. *Computer Graphics*, 20(4):143–150, 1986.

[30] D S Kay and D Greenberg. Transparency for computer synthesized images. *Computer Graphics*, 13(2):165–174, 1979.

[31] D Kirk and J Arvo. Unbiased sampling techniques for image synthesis. *Computer Graphics*, 25(4):153–156, 1991.

[32] B Lange. The simulation of radiant light transfer with stochastic ray tracing.

[33] M E Lee, R A Redner, and S P Uselton. Statistically optimized sampling for distributed ray tracing. *Computer Graphics*, 19(3):61–65, 1985.

[34] G Meyer. Colorimetry and computer graphics. *Program of Computer Graphics, Cornell University, Ithaca, NY*, 1983.

[35] G Meyer. Wavelength selection for synthetic image generation. *Computer Vision, Graphics, and Image Processing*, pages 57–79, 1988.

[36] S P Mudur and S N Pattanaik. Monte Carlo methods for computer graphics. *Eurographics 93, State of the Art Report*, pages 1–24, 1993.

[37] S N Pattanaik. *Particle Tracing with the Potential Equation*. PhD thesis, University of Cornell, 1992.

[38] K Perlin. An image snthesizer. *Computer Graphics*, 19(3):287–296, 1985.

[39] B T Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[40] L Piegl. On NURBS: A survey. *IEEE Computer Graphics*, pages 55–71, 1991.

[41] R J Recker, D W George, and D P Greenberg. Acceleration techniques for progressive refinement radiosity. *Computer Graphics*, 24(2):59–66, 1990.

[42] S M Rubin and T Whitted. A three dimensional representation for fast rendering of complex scenes. *Computer Graphics*, 14(3):110–116, 1980.

[43] T Setzer. Internet newsgroup.

[44] R Siegel and J R Howell. *Thermal Radiation Heat Transfer*. McGraw Hill, 1981.

[45] F Sillion and C Puech. A general two-pass method integrating specular and diffuse reflection. *Computer Graphics*, 23(3):335–344, 1989.

[46] S N Spencer. The hemisphere radiosity method: A tale of two algorithms. *Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 127–135, 1990.

[47] K E Torrance and E M Sparrow. Theory for off-specular relection from roughened surfaces. *Journal of the Optical Society of America*, 57(9):1105–1114, 1967.

[48] S Upstill. *The Renderman Companion*. Addison Wesley, 1990.

[49] G J Ward, F M Rubinstein, and R D Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics*, 22(4):85–92, 1988.

[50] D R Warn. Lighting controls for computer graphics. *Computer Graphics*, 17(3):13–21, 1983.

[51] T Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 26(6):343–349, 1980.