LEABHARLANN CHOLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH
Ollscoil Átha Cliath

TRINITY COLLEGE LIBRARY DUBLIN
The University of Dublin

Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Monte Carlo Simulation for Global Illumination

William Leeson

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science
University of Dublin, Trinity College
Department of Computer Science
February 23, 2001

Acknowledgements

I would like to thank Cinty Broadhead and Mark "Sparks" Dennehy and my parents without whose help this thesis would never have been understandable. Also I would like to thank my supervisor Carol O'Sullivan for putting up with me for the last 2 years and helping to make my papers more readable. Thanks also go to my original supervisor Steven Collins without whom I would never have been able to get into global illumination in the first place. Finally my lab mates Leo Talbot and Gareth Bradshaw for the odd frag fest at lunch time and their invaluable criticism without which I would not have strived to do better. I would also like to thank my parents again for putting up with me for the last 4 years when I should have been working at a "real" job earning money.

Summary

The aim of global illumination is to produce a realistic image that is indistinguishable from a real image of that scene in the least amount of time. This is achieved by modelling the physical laws that govern light propagation. Global illumination effects have been used in films and computer games to create more immersive environments. Unfortunately, the time taken to render a physically correct image is far higher than with more conventional three dimensional computer graphics. This not because the processes are very different but because more evaluations of certain operations such as visibility determination are required. The high computational costs can be addressed by either improving the efficiency of the algorithms or by parallelising them. Many of the current global illumination algorithms are limited in the types of scenes they can render and the surface models that can be used. Usually an importance sampling function has to be designed for each surface model. Without this function the algorithm will not perform well. In addition, as scenes get more complex, the performance of some algorithms begins to falter.

The principal contributions of this work are a framework which uses mathematical concepts, two different adaptive integration methods are use to solve the rendering and potential equations, a parallel rendering scheme is devised which uses stratified sampling, a Metropolis sampling method is created which uses density estimation rather than integration and finally a scheme for compressing the particles produced by the Metropolis sampling method.

This thesis first presents a modular, extendible, flexible and object oriented framework which is able to cope with the many different types of global illumination algorithms. The framework was designed so that many algorithms could be added with relative ease. It is based on mathematical concepts such as integration and function evaluation, which are common to all global illumination methods.

Monte Carlo techniques have been applied in the past to the problem of global illumination. However, many previous techniques have been unable to cope with a large range of scene types and surface representations. To improve this situation, two numerical integration algorithms have been modified and used to render images. Two adaptive algorithms are proposed: One is derived from the VEGAS algorithm which uses importance sampling, and the second is based on stratified sampling.

In order to speed up the rendering of scenes, a generic and flexible parallel algorithm has been devised which uses stratification to break up the problem domain. This enables it to be used without modification with the new adaptive algorithms, among others. By changing the parameterisation of the problem domain, the partitioning can be used to create greater coherency and also to facilitate the rendering of large scenes.

Finally, Metropolis sampling is applied in a novel way to distribute photons, either on the surfaces of the scene or on the film plane, using the potential or rendering equation. By using density estimation methods on these samples the illumination is reconstructed. Due to the correlation present in the samples produced by the Metropolis algorithm, the particle stream may be compressed. To aid compression the particle data is rearranged so that compressible regions are more readily identifiable.

Declaration

I declare that the work described in this thesis has not been submitted for a degree at any other university, and that the work is entirely my own.

Signature

[illegible]

William Leeson

Permission to lend and/or copy

I agree that the library in Trinity College may lend or copy this thesis upon request

Signature

[illegible]

William Leeson

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis deals with the subject of Global Illumination. Its goal is to produce a two dimensional image that appears realistic to the viewer, given adequate input data which describes the geometric and surface properties of the scene and viewing device. By using the physical laws governing light propagation more realistic imagery can be produced. The more physically correct an illumination model is the more likely that it will appear realistic to a viewer. In its earliest use only simple shading models and hidden surface algorithms were used, these featured in films such as "Tron" (see Figure 1.1) and "The Last Star Fighter". Todays blockbuster films such as "The Abyss" which won awards for its special effects, "Star Wars", "Titanic" and "The Matrix" all use more sophisticated techniques to create stunning effects. Thanks to early pioneering companies such as Newtek, even low budget TV shows such as Babylon 5 and Star Trek feature stunning visual effects. Pixar has led the way in computer animation with its well known animation "Luxo Jr." which earned many awards and for its feature film "Toy Story" which was a box office hit. These films were produced using their Renderman package.

Today nearly all films feature computer graphics effects to some degree. Even computer games feature realistic imagery which is used to better immerse the player in the virtual environment. Computer games such as Quake and Unreal use a global illumination technique called radiosity to pre-calculate shadows and lighting. Today many rendering packages are available to the public, some of which are free. This has increased awareness and interest in rendering and global illumination and as a result a demand for ever better and more realistic images has arisen. People are starting to make home grown movies known as Machinima, using game engines such as Quake where the characters are actors controlled by players. This is becoming popular as people can produce virtual movies on cheap PCs. New software is being developed by a number

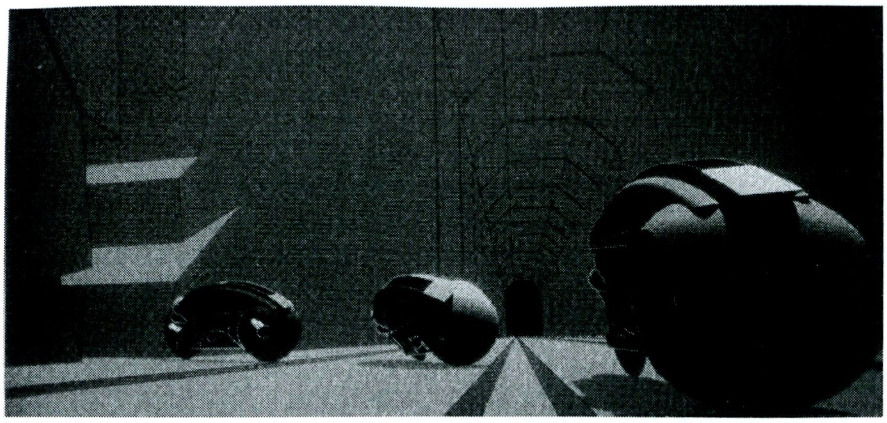


Figure 1.1: Early computer graphics from the Film Tron

of companies specifically to cater for this type of film production. Many of the cartoons appearing on TV today are done entirely using computer graphics such as Star Ship Troopers and ReBoot. However, to date very few of these productions use a full global illumination capable rendering system. This makes it relatively easy to distinguish the difference between scenes using computer graphics and those without. This is mainly due to the speed and memory costs of global illumination techniques which at the moment are impractical for large animations. New possibilities are created by global illumination and the adoption of these methods to create an ever more believable environment is inevitable.

1.1 Stages Of Rendering

Global Illumination is different from most of the other fields in computer graphics in that it uses fixed models which must be solved. The problems involved bring together many different subjects including programming, mathematics and psychology in order to solve what initially appears to be a simple problem. Early realistic graphics research primarily dealt with local illumination, which was a visibility problem i.e. "Can I see the light from this point?" or "Can the viewer see this point?" Global Illumination is an extension of this concept to - "Can I see this object via reflection or refraction from another object?" This is however a very simplified view of global illumination. The complete process can be split into three stages

- **Modelling:** This is the first stage in any image synthesis process. A three dimensional scene must be constructed. This is done by specifying the geometric models of objects, along with their reflective and refractive surface properties and textures. Lights are added to the scene and also cameras whose configurations can be specified in a number of ways. The specifications can be entered in a variety of forms, usually with the aid of a modelling program. This stage is very important as badly designed scenes can also impair the visual realism of the image produced.

- **Rendering:** Early attempts at rendering dealt with wire frame or solid filled representations, where the main goal was hidden surface removal. Photo realistic image synthesis deals with the solution of complex equations. This can be split into two parts: A view independent stage and a view dependent stage. The first deals with the construction of the lighting in a three dimensional space which can be later viewed using a view dependent method. View dependent methods deal with the construction of a 2D image given the position of the viewer. They can use information from the view independent stage or construct a solution independently. There are a variety of rendering algorithms that can be used to produce images but most are only suitable for solving limited lighting situations and can be very slow depending on the accuracy and level of realism used.

- **Tone Reproduction:** This involves converting the physical quantities used to represent the light incident on the film plane into an image that a human observer would perceive from the same data or input in the real world. This stage to some extent leaves the area of physics and starts to deal with the psychology of perception.

In any image generation program these steps can be combined. In fact the final two stages are usually mixed to some extent, as the Tone Reproduction helps to point out the importance of regions in the image plane with respect to a human observer. This helps to lower the computational demands which can be very high for some methods.

1.2 Applications

Photo realistic image synthesis has many potential uses in industry.

- **Films and TV:** Recently many films have featured state of the art photo realistic effects. This is an area of particular promise for global illumination as the goal of a film is to immerse the viewer in a fictitious world and this is an area where computer graphics excel. Computer graphics can be used to visualise scenarios which are either too expensive or impossible to create in reality. One of the first TV programs to feature realistic computer generated imagery was Babylon 5, a science fiction program.

It featured space battles consisting of hundreds of ships rather than the limited numbers that could be achieved with conventional model-based motion control techniques.

- **Architecture:** The design of buildings is very important. To assess the aesthetics usually an artist's representation of the building is used. This method unfortunately can be very misleading. The use of global illumination to synthesise photographs of the building is a more accurate and believable approach. Currently a few rendering packages are available to assess the aesthetics of a building, one such package is Lightscape.

- **Lighting:** The optimal placing of lights and windows in a building is very important. Global illumination can be used to help maximise the contribution of a light to a scene and also to enhance the natural lighting of a building, by testing various configuration of lights. The object is to minimise the number of lights as well as making them more effective and therefore decreasing the cost of lighting the building. Another area of use is in designing stage lighting to create dramatic sets for theatres and films.

- **Games:** Companies are striving to achieve ever more realistic and impressive imagery for use in games. The more realistic a game appears the more readily the viewer gets immersed in the game, the ultimate goal being to produce cinematic quality imagery. Lighting and global illumination effects are used to create atmosphere. Some of the most immersive experiences in games involve using lighting effects, for instance switching off the lights, making low intensity lighting settings or using fog.

- **Art:** Computers recently have been used to create amazing pictures and animations. The realistic imagery made possible with global illumination could also be used to produce art.

- **Others:** There are many possible uses of realistic imagery, such as in virtual reality flight and driving simulators for training, and many more.

Moore's law predicts that the power of computers double every 18 months and this still holds. Thus, today's rendering algorithms which were once too slow to be practical can now be run by any off-the-shelf computer. It is therefore conceivable that the global illumination algorithms of today which are too slow for today's computers will become practical in a few years. In fact current games now feature global illumination techniques to a limited extent. Ray tracing once the slowest of rendering methods can now be done in real time [87, 88]. Recently, 3D acceleration hardware has undergone a dramatic upheaval, and what was once only available in the domain of the professional is now available to all cheaply. This has produced a massive growth in innovations and advances of realistic imagery and some of this hardware can be used to accelerate global illumination problems.

1.3 Objectives and scope

The aim of this thesis is to produce global illumination algorithms which are:

- **physically correct:** Able to represent and produce physically correct data
- **robust:** Able to solve all situations
- **efficient:** Fast at producing results with low memory overheads
- **effective:** Able to create accurate estimates in the minimum time.

Another goal is to produce a framework for construction of these algorithms which is:

- **intuitive:** easy for both the user and developer to use
- **fast:** at producing images and at constructing rendering systems
- **expandable:** to the extent that new methods and techniques can be added easily
- **modular:** so that each component is separate and can be used interchangeably.

1.4 Thesis Overview

Chapter 2 Problem Description This chapter describes the problem domain and various ways used to describe light transport and other associated concepts using mathematical models. To achieve this the physics of light and light transport are described. Then a number of models used in global illumination are described in detail.

Chapter 3 Problem Solution Reviews the mathematical methods used to help solve the problems presented in the previous chapter. A description of methods for solving integral equations and density estimation methods are included.

Chapter 4 Parallel Methods A review of parallel methods and terminology is presented. Details of various concepts and types of scheduling algorithms are described with their potential advantages and disadvantages. The final part includes a review of current parallel techniques used in global illumination.

Chapter 5 Algorithms A number of new algorithms and the framework used to construct them are presented. The design, construction and reasons for the development of these are explained. A series of adaptive algorithms are presented which use integration to solve the global illumination problem. Next a parallel method is used to accelerate the previous methods as well as some particle tracing methods. Finally density estimation and Metropolis methods are described which can be used in conjunction with the adaptive methods or on their own to solve global illumination scenarios.

Chapter 6 Results An analysis of the algorithms described in the previous chapters in terms of efficiency, effectiveness and memory consumption is provided.

Chapter 7 Conclusions and Future Work Here the main ideas presented in the thesis are summarised and potential areas for further investigation and research are highlighted.

Chapter 2

Problem Description

Global illumination is the physical simulation of light in an environment, with the goal of producing a synthetic image of a scene that appears identical to that which an observer in the real world would experience. Unfortunately, representing the whole experience is not possible in all situations and may not always be desirable, hence the image a camera would record is often the preferred goal. To properly simulate the lighting in a room, the entire environment of the room must be taken into account. To obtain this image both the physical aspects involved and the psychology of perception need to be taken into account. To calculate the amount of light reaching a point in space, a model must be created which is capable of describing the physical process in sufficient detail such that all the desired effects can be simulated. It was not until about 1984 with the introduction of the radiosity methods by researchers in Fukuyama and Hiroshima Universities [86] in Japan and at the Program of computer graphics at Cornell University [45] in the United States, that physically correct global illumination models were developed. These methods were restricted to having only diffusely reflecting surfaces. Later, with the introduction of the rendering equation [63], less restrictive global illumination methods became possible. In order to generate a realistic image of an environment as seen through a camera, a mathematical model of how light interacts with an environment is needed. What follows is the construction of a simple model of light that describes its interaction with an environment. It uses some physics to describe light and those properties which are relevant to the effects we wish to reproduce. These concepts are then represented in mathematical equations which we will use later. The following discussion is based on that in "Radiosity and Realistic Image Synthesis" [19].

2.1 Modelling the Objects

Modelling objects in a scene, such as chairs and desks, is done using geometry such as polygons, spheres and parametric patches. Using combinations of these primitives, complex scenery can be built up. By assigning properties to the surfaces, a sufficiently accurate representation of many real world objects can be created. Using this representation and a model for the interaction of light with this environment the illumination of any surface in the scene may be estimated.

2.2 Basic Optics

Light is a form of electromagnetic radiation, formed by coupled electric and magnetic fields which are perpendicular to the direction of propagation and which form a sinusoidal wave (see Figure 2.1). The frequency of oscillation of this wave determines its wavelength. These waves can have many properties, for instance light may be:

polarised meaning that the electric and magnetic fields may have a single orientation;

coherent meaning that the light contains a fixed frequency.

Light can be described or modelled on three different levels:

geometrical or ray optics model light at a basic level of reflection and refraction;

wave or physical optics describe light effects such as diffraction, interference and dispersion. This model describes light interaction with objects of sizes near the wavelength of light;

quantum or photon optics model light using photons which are particles that have both wave and particle properties given by ray and wave optics respectively.

Geometrical optics is the most suitable model for use in global illumination because it deals with light at the macroscopic level. This assumption is very important, as it limits the optical phenomena we can model. This is because geometrical optics cannot handle effects such as interference, diffraction and dispersion, for this we need wave optics.

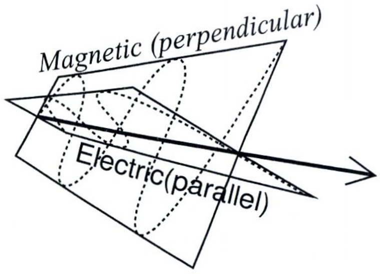


Figure 2.1: Light Wave

2.3 Radiometry

Radiometry is the science of physical measurement of electromagnetic energy. The radiometric unit of energy is the *joule* and for *power* is the watt (equivalent to joules per second).

2.3.1 Energy Conservation

Conservation of energy is an important principle that must be taken into account in any light transport model. Energy must be conserved on two scales:

globally The total power radiated into a system by light sources must equal that absorbed by objects. This absorbed energy flows out as heat energy in most cases.

locally Energy flowing into a region of space must equal that leaving the region.

2.3.2 Solid Angle

The angle subtended by a circular arc of length l is equal to l/r, this is known as a *radian*. Using a similar concept, the idea of an angle can be extended into the three dimensional world to get a/r^2 (see Figure 2.2) where a is the area of a patch on the surface a unit sphere. The units of this quantity are called *steradians* or radians squared. A differential solid angle $d\omega$ is thus

$$d\omega = \frac{dA}{r^2} = \sin\theta d\theta d\phi$$

2.3.3 Units and Notation

Before we can quantitively discuss how light intraction is modeled some symbols and units need to be defined. The symbols used for the mathematical notation

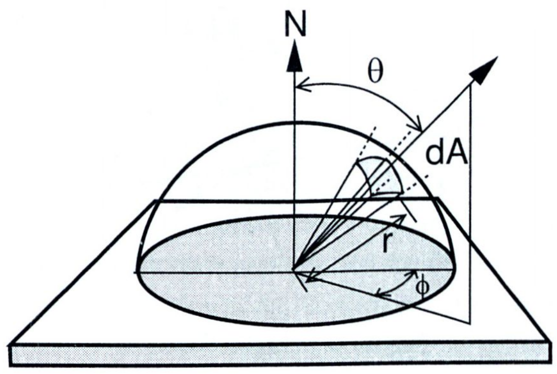


Figure 2.2: Solid Angles

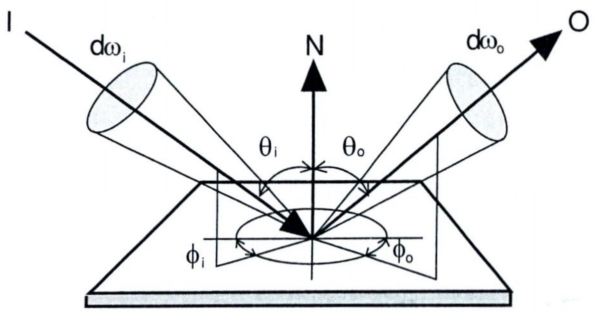


Figure 2.3: Directions

are given in Table 2.1 and the units in Table 2.2. The diagram in Figure 2.3 shows the concepts the symbols relate to.

2.3.4 Radiance

In the case of rendering systems the quantum nature of light is usually not considered. Instead the radiant energy per unit time, or radiant power Φ, rather than the number of particles, is used as the flux. This value is the photon volume density times the energy of a single photon and is called the *radiance* L. Thus radiance is the power per unit projected area perpendicular to the ray per unit solid angle in the direction of the ray, and thus its units are $Wm^{-2}sr^{-1}$.

2.3.5 Radiant Flux Density

The radiant flux can be considered as either incoming or outgoing. When incoming it is called the *Irradiance* E which is the radiant energy per unit area incident onto a surface with a fixed orientation. It is calculated by integrating

A	area
x	position
ω	solid angle
θ	zenith angle
ϕ	azimuth angle
E	irradiance
L	radiance
B	radiosity
λ	wavelength
k_{x_λ}	diffuse coefficient x for wavelength λ

(a) Symbols

i	incoming
o	outgoing
r	reflected
t	transmitted
d	diffuse
s	specular

(b) Subscripts

Table 2.1: Notation Tables

W	Watts
m	metres
sr	steradians

Table 2.2: Table of Units

the incoming radiance L_i over a hemisphere Ω_i as follows

$$E = \int_{\Omega_i} L_i \cos\theta_i d\omega_i \tag{2.1}$$

$$= \frac{d\omega_i}{dA} \tag{2.2}$$

Where $\cos\theta_i d\omega_i$ defines a projected area on the base of the sphere. The outgoing radiant flux density, called the *Radiosity* B, is the inverse of the irradiance term. Radiosity is defined as the energy per unit area that leaves a surface. Thus it can be found also by integrating over the hemisphere. This time we integrate outgoing radiance L_o over the hemisphere Ω_o

$$B = \int_{\Omega_o} L_o \cos\theta_o d\omega_o \tag{2.3}$$

$$= \frac{d\omega_o}{dA} \tag{2.4}$$

Both radiosity and irradiance can be expressed as the energy per unit area with units of Wm^{-2}.

2.4 Modelling Reflection and Transmission

Describing reflection and transmission of light from a surface is an important consideration in any illumination model. For a complete description of light interaction at a surface you would need to take into account florescence, polarisation, phosphorescence and many more effects. However in order to simplify these models, simple geometrical optics are generally used.

2.4.1 Reflection and Refraction

Reflection is defined as the process by which light incident on a surface leaves that surface from the same side. Refraction being the opposite process by which light incident on a surface leaves that surface from the opposite side. Materials can be categorised into two distinct types, *conductors* and *insulators.* The conductors are said to absorb light energy and are characterised by complex indices of refraction, whereas insulators do not absorb any light energy and they are characterised by non-complex indices of refraction. Conductors are non-transparent, examples of such materials are metals; insulators on the other hand are generally transparent substances such as glass.

The Bi-Directional Reflection Distribution Function BRDF

From experimentation, it has been observed that the light reflected L_r from a surface is proportional to the light incident $E(\vec{\omega}_i)$ on that surface:

$$dL_r \propto dE(\vec{\omega}_i)$$

The *bi-directional reflection distribution function* (BRDF) describes the reflection of light from a surface, and is given by the following:

$$f_r(\vec{\omega}_i \to \vec{\omega}_r) = \frac{L_r(\vec{\omega}_r)}{L_i(\vec{\omega}_i)\cos\theta_i d\omega_i} \tag{2.5}$$

where $\vec{\omega_i}$ is the incoming solid angle, $\vec{\omega_r}$ is the reflected solid angle and L represents the corresponding radiances and θ_i is the angle between the surface normal and the incoming solid angle (see Figure 2.4). The BRDF is therefore the ratio of incident radiance in a given solid angle to the reflected radiance. The BRDF may have some of the following properties:

1. The Helmholtz reciprocity principle: If based on physical laws, the BRDF will remain unchanged if the incoming and outgoing directions are reversed.

2. Anisotropic: If the incoming and outgoing directions are fixed and the surface rotated, then the percentage of reflected light may change.

3. Isotropic: If the incoming and outgoing directions are fixed and the surface rotated, then the percentage of reflected light will not change.

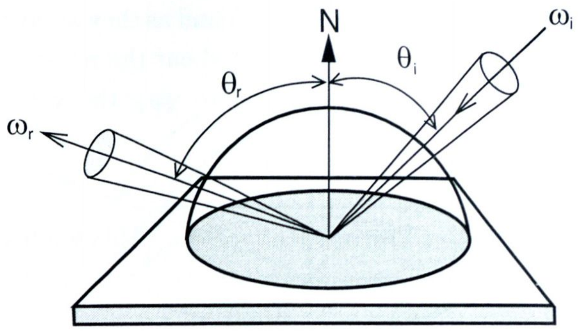


Figure 2.4: BRDF parameters

What follows are a few examples of BRDFs which are commonly used in rendering systems. For the purposes of simplicity and clarity the wavelength dependence of these functions will be ignored. In order to have the functions vary according to wavelength λ the coefficients k_{x_λ} can be made a function of wavelength.

BRDF Model for Perfect Reflection The properties of a perfect reflector are

- $\theta_r = \theta_i$
- $\phi_r = \phi_i \pm \pi$
- $L_r(\theta_r, \phi_r) = L_i(\theta_r, \phi_r \pm \pi)$

where the subscripts i and r represent incoming and outgoing directions and θ and ϕ are from the spherical coordinate system introduced previously. That is to say, the reflected vector ($[\theta_r, \phi_r]$) is in the plane given by the incident vector ($[\theta_i, \phi_i]$) and the normal to the surface, and the incoming radiance (L_i) is equal to the outgoing radiance (L_r). Thus the BDRF can be expressed as

$$f_r(\vec{\omega}_i \to \vec{\omega}_r) = k_{s_\lambda} \frac{\delta(\cos\theta_i - \cos\theta_r)}{\cos\theta_i} \delta(\phi_i - (\phi_r \pm \pi)) \tag{2.6}$$

Where k_{s_λ} is the portion of reflected light. Note that this BRDF uses the delta function $\delta(x)$, which has the following properties:

- $\delta(x) = 0$ if $x \neq 0$
- $\int_{-\infty}^{\infty} \delta(x)dx = 1$
- $\int_{-\infty}^{\infty} \delta(x - y)f(x)dx = f(y)$

In nature however, perfect mirrors are never found as they always contain imperfections which causes the rays be distributed about the reflected ray. Normally the Fresnel formulae for reflection are used to vary the $k_s\lambda$ term, these are discussed in Appendix A

BRDF Model for Perfect Diffuse Reflection This is a model which scatters light uniformly in all directions and is a constant given by

$$f_r(\vec{\omega}_i \to \vec{\omega}_o) = \frac{k_{d_\lambda}}{\pi} \tag{2.7}$$

where k_d is a coefficent which describes the portion of light reflected by the surface. As in the model for perfect reflection, perfectly diffuse surfaces do not occur in nature.

Phong BRDF Model for Glossy Reflection This is a BRDF which is based on the original Phong shading model [97]. Although the original implementation of the model is not a valid BRDF (as physical constraints were not taken into its design) , Lafortune and Willems [71] have created a valid BRDF model which gives similar looking surface finishes.

$$f_r(\vec{\omega}_i \to \vec{\omega}_o) = \frac{k_{d_\lambda}}{\pi} + k_{s_\lambda}\frac{n+2}{2\pi}\cos^n\alpha \tag{2.8}$$

α is the angle between the perfect specular reflective direction and the outgoing direction. Values larger than $\pi/2$ should be clamped, to avoid getting any negative cosine values. k_{s_λ} and k_{d_λ} are the specular and diffuse coefficients and n the specular exponent. Energy conservation is guaranteed if and only if $k_{d_\lambda} + k_{s_\lambda} \leq 1$.

Ward BRDF Model for isotropic and anisotropic reflection The Ward BRDF model [134] is a more physically plausible model which can be used to model anisotropic as well as isotropic surfaces. The isotropic model is a simplification of the anisotropic model and is given by

$$f_r(\vec{\omega}_i \to \vec{\omega}_o) = \frac{k_{d_\lambda}}{\pi} + k_{s_\lambda}\cdot\frac{1}{\sqrt{\cos\theta_i\cos\theta_r}}\cdot\frac{\exp(-\tan^2(\frac{\theta_h}{\alpha^2}))}{4\pi\alpha^2} \tag{2.9}$$

where α is the standard deviation (RMS) of the surface slope and θ_h is the zenith angle of the vector halfway between the incoming θ_i and outgoing θ_o rays (see Figure 2.5). In order to ensure energy conservation, $k_{d_\lambda} + k_{s_\lambda} \leq 1$; and the normalisation factor $1/4\pi\alpha^2$ is accurate as long as α is not much greater

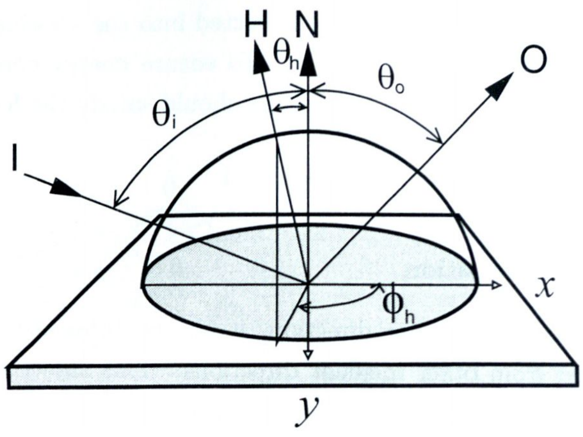


Figure 2.5: Halfway Vector

than 0.2. The full model which supports anisotropic scattering is as follows

$$f_r(\vec{\omega}_i \to \vec{\omega}_r) = \frac{k_{d_\lambda}}{\pi} + k_{s_\lambda}.\frac{1}{\sqrt{\cos\theta_i \cos\theta_r}}.\frac{\exp(-tan^2\theta_h(\cos^2(\frac{\phi_h}{\alpha_x^2}) + \sin^2(\frac{\phi_h}{\alpha_y^2}))}{4\pi\alpha_x\alpha_y} \tag{2.10}$$

where α_x and α_y are the standard deviation of the surface slope in the x and y directions at the tangent plane and ϕ_h is the azimuth angle of the halfway vector (see Figure 2.5). In order to maintain energy conservation we require that $k_{d_\lambda} + k_{s_\lambda} \leq 1$ and that $\alpha_x^2 \ll 1$ and $\alpha_y^2 \ll 1$.

Bi-directional Transmission Distribution Function (BTDF)

The definition of this function is exactly the same as that of the BRDF, except that this quantity describes the transmission of light through a surface instead of reflection from it. Thus the reflection term L_r is replaced by a transmission term L_t in Equation 2.5

$$f_t(\vec{\omega}_i \to \vec{\omega}_t) = \frac{L_t(\vec{\omega}_t)}{L_i(\vec{\omega}_i)\cos\theta_i d\omega_i} \tag{2.11}$$

Bi-directional Scattering Distribution Function (BSDF)

This is a combination of both the BRDF and the BTDF. By combining these two quantities, a complete description of all scattering events at an interface is obtained. Usually a BSDF is created by just combining the formulae for a BRDF and a BTDF. This is typically done for the diffuse and Phong models as follows

$$f(\vec{\omega}_i \to \vec{\omega}_o) = \alpha_r f_r(\vec{\omega}_r \to \vec{\omega}_r) + \alpha_t f_t(\vec{\omega}_i \to \vec{\omega}_t) \tag{2.12}$$

where ω_r is the incoming light that is reflected into the viewing direction and ω_t is the corresponding refracted light. To ensure energy conservation. the reflected α_r and transmitted α_t coefficients should satisfy the following:

$$\alpha_r + \alpha_r \leq 1 \tag{2.13}$$

The Reflectance Equation

If light is added from another direction, it has no influence on the amount of light reflected from other incident directions. This shows that reflectance behaves linearly, and so the total amount of light reflected by a surface in a specific direction is given by an integral on the hemisphere over all possible incident directions. This ignores non-linear effects such as those produced by laser light, these effects are modelled by wave and quantum optics which are beyond the scope of this dissertation. The *Reflectance Equation* which describes the amount of light that is reflected through a solid angle is given by

$$L(\vec{\omega}_i \rightarrow \vec{\omega}_o) = \int_\Omega f(\vec{\omega}_i \rightarrow \vec{\omega}_o) L_i(\vec{\omega}_i) \cos\theta_i d\omega_i \tag{2.14}$$

Note the reflectance equation and the rendering equation which follow rely on the concept of integral equation which are discussed in detail in Appendix B.

2.5 Rendering Equation

The light leaving a surface consists of two parts, the emitted light and the reflected or transmitted light. In general, the radiance of a surface is a function of both the position and viewing direction. Assuming that the geometry, emissive, reflective and transmissive properties of all surfaces are known, the radiance at each surface point in each direction can be determined. The light scattering at a surface point in a given direction could, in general, have come from any direction. Some of this light will come from the light sources and the remaining light will come from other surfaces (see Figure 2.6). The radiance at a point is related to the radiance of all points visible to it. The equation describing these interdependencies has the general form

$$\text{radiance}(y) = \text{emit}(y) + \int_a$$

$$\text{radiance}(y) = \text{emit}(y) + \int_{\text{all surfaces}} \text{radiance}(x)\text{scat}(y, x) dx \tag{2.15}$$

where y and x each represent a point and a direction, radiance(y) is the outgoing radiance at y, emit(y) is the emitted radiance at y and scat(y, x) is the fraction of light between points y and x. This, as can be seen, is an integral equation

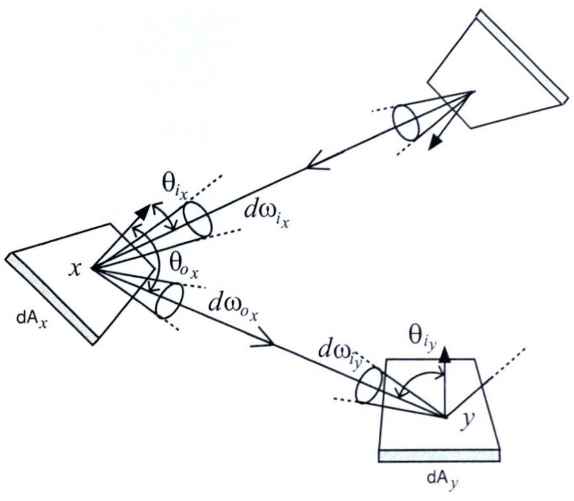


Figure 2.6: Indirect Lighting

because the unknown function radiance() appears both outside and inside the integral.

In order to derive a global illumination model, occlusion by, as well as light reflected from other surfaces must be considered. Therefore in the equation, radiance on one surface must be related to that on another:

$$L_i(y, \vec{\omega}_{i_y}) = L_o(x, \vec{\omega}_{o_x}) V(x, y)$$

This relates the incident radiance on one surface at y to the radiance from x. Where ω_{i_y} is the direction vector from y to x and ω_{o_x} is in the opposite direction. The function $V(x, y)$ is the visibility function,

$$V(x,y) = \begin{cases} 1 & \text{if } x \text{ is visible from } y, \\ 0 & \text{otherwise.} \end{cases}$$

Relating a integral over all directions to an integral over all surfaces is done by relating the solid angle subtended by the source to its projected surface area

$$d\omega_{i_x} = \frac{\cos\theta_{i_x} dA_x}{|x - y|^2}$$

Converting to a projected solid angle gives

$$d\omega_{i_x} \cos\theta_{o_x} dA_x = G(x, y) dA_x$$

Letting

$$G(x,y) = G(y,x) = \frac{\cos\theta_{i_x}\cos\theta_{o_y}}{|x-y|^2}$$

and substituting into the reflectance equation yields

$$L(y,\vec{\omega}_{i_y}) = \int_S f(\vec{\omega}_{i_x} \rightarrow \vec{\omega}_{o_x})L(x,\vec{\omega}_{i_x})G(x,y)V(x,y)dA_x$$

Then finally taking into account the energy of the point on the surface,

$$L(y,\vec{\omega}_{i_y}) = L_e(y,\vec{\omega}_{i_y}) + \int_S f(\vec{\omega}_{i_x} \rightarrow \vec{\omega}_{o_x})L(x,\vec{\omega}_{i_x})G(x,y)V(x,y)dA_x \qquad (2.16)$$

This however, is not the only form of the rendering equation. There are many others. In fact this version is usually converted into a integral over all directions

$$L(y,\vec{\omega}_{i_y}) = L_e(y,\vec{\omega}_{i_y}) + \int_{\Omega_i} f(\vec{\omega}_{i_x} \rightarrow \vec{\omega}_{o_x})L(x,\vec{\omega}_{i_x})(-\vec{\omega}_{i_x}.\vec{n})d\omega_{i_x}) \qquad (2.17)$$

This is not a complete light transport model. Effects such as participating media, where light is scattered and therefore attenuated as it travels through a medium, are beyond the scope of this thesis.

2.6 Potential Equation

Due to the optical properties of different surfaces, the light emitted from one surface in a given direction can illuminate many other surfaces. To be able to estimate this quantity, the potential equation [92, 89, 90] was introduced. Progressive radiosity [18] and particle simulation [94, 93, 91, 15] are two methods which use the potential equation. The potential equation is an adjoint of the rendering equation and is based on the propagation of light starting from the light sources. It expresses the potential capability W of any point and direction towards the illumination of another, this is expressed as follows:

$$W(y,\vec{\omega}_{o_y}) = g(y,\vec{\omega}_{o_y}) + \int_{\Omega_o} f(\vec{\omega}_{o_x} \rightarrow \vec{\omega}_{i_x})W(x,\vec{\omega}_{o_x})cos\theta_{o_x}d\omega_{o_x} \qquad (2.18)$$

Equation 2.18, although it looks very similar to the rendering equation 2.17, is not the same. The difference is that in the rendering equation y is the point visible to x along an incoming direction at x and integration is over the incoming hemisphere about x (i.e. the direction of the rays). The function $g(y,\omega_{o_y})$ is crucial to the exact meaning of the potential. When $g(y, \omega_{o_y})$ is as follows:

$$g(x, \omega_{o_y}) = \begin{cases} cos\theta_{o_y} & \textit{iff}\, x \in \text{Surface}_k \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

then $W(x, \omega_{o_y})$ defines the potential ability of (y, ω_{o_y}) to contribute to the total outgoing flux of the surface k in the environment. If on the other hand $g(y, \omega_{o_y})$ is given as:

$$g(y, \omega_{o_y}) = \begin{cases} \frac{1}{dA_x d\omega_{o_y}} & \textit{iff}\, (x, \omega_{o_y}) = (z, \omega_z), \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

$W(y, \omega_{o_y})$ will now express the potential capability of (y, ω_{o_y}) to contribute towards the radiance at (z, ω_z).

2.7 Other work

There have been many alternative models for describing light transport, most of which are based on the potential and rendering equations given above. The Global Reflection Distribution Function (GRDF) [32] is a reformulation of both equations and is used to describe the hypothetical light transport between two arbitrary points and their associated directions in a three dimensional environment. This allowed the author to create a bi-directional path tracing [69, 73] implementation. Veach created another formulation based on ray paths to describe his alternative bidirectional path tracing algorithm [129] and also the Metropolis Light Transport algorithm [130] using a transport operator [128] which made symmetry and reciprocity in BRDFs the same thing.

There have also been many different BRDFs created that are more sophisticated than those described previously. A number of surveys of many BRDFs in common use are available [113, 98, 116]. New BRDFs for metals are given by Kalos in [83], a formulation for a general BRDF was given by Neumann in [84], and another general BRDF formulation using non-linear parameters was given by Lafortune *et al* [74]. A method of representing BRDFs using wavelets was presented by Lalonde [75], and another discrete approximation using a geodesic sphere was shown by Gondel, Meyer and Newmann [44]. More recently a micro facet [8] based BRDF representation scheme has been proposed.

The number of variations described here serve to show the diversity of techniques and methods used to describe many of the models used in light transport. Each technique has it own advantages and corresponding limitations as a result of the assumptions made in creating the model.

Chapter 3

Methods of Solution

This chapter presents a summary of the myriad of Monte Carlo and density estimation techniques that may be used to solve global illumination problems, using the equations presented in the previous chapter. These methods provide a means of solving many of the problems described by the models in the last chapter. The first section starts by introducing Monte Carlo integration methods next the Markov Chain Monte Carlo method is described, this can be used to obtain samples according to a given function. This is followed by an explanation of how these methods can be used on the models described in the previous chapter. Finally density estimation methods are detailed.

3.1 Monte Carlo Methods

A definition of a Monte Carlo method would be one that involves deliberate use of random numbers in a calculation that has the structure of a stochastic process. A *stochastic* process is a sequence of states whose evolution is determined by random events.

3.1.1 Monte Carlo Integration

This section requires an understanding of some elementary probability theory covered in Appendix C. Given a set of random variables $x_1, x_2, x_3, \ldots, x_n$ taken at random from the PDF $f(x)$, a function G may be defined as

$$G = \sum_{n=1}^{N} \lambda_n g_n(x_n) \tag{3.1}$$

where $\lambda_n \in \mathbf{R}$ and g_n may be a different function of x_n. Since each g_n is a random variable, therefore $\sum g_n(x)$ is also random. Thus the expectation of G is given by

$$E(G) = \langle G \rangle = E\left(\sum_{n=1}^{N} \lambda_n g_n(x_n)\right) \tag{3.2}$$

Since the expectation is a linear operation this can become

$$E(G) = \sum_{n=1}^{N} \lambda_n \langle g_n(x_n) \rangle \tag{3.3}$$

If all the x_n are independent then the variance of G is

$$\begin{aligned} \mathrm{var}\{G\} &= \langle G^2 \rangle - \langle G \rangle^2 \\ &= \sum \lambda_n^2 \mathrm{var}\{g_n(x)\} \end{aligned} \tag{3.4}$$

If all the $g_n(x)$ are identical and $\lambda_n = 1/N$ then the expectation of G becomes the following

$$\begin{aligned} E(G) &= E\left(\frac{1}{N}\sum_{1}^{N} g(x_n)\right) \\ &= \langle g(x) \rangle \end{aligned} \tag{3.5}$$

Now the function G is the average of $g(x)$. G is said to be an *estimator* of $\langle g(x) \rangle$. In more general terms, a function G is an estimator of a quantity if its mean $\langle G \rangle$ is a usable approximation of that quantity. The variance of G is

$$\begin{aligned} \mathrm{var}\{G\} &= \mathrm{var}\left\{\frac{1}{N}\sum_{1}^{N} g(x_n)\right\} \\ &= \sum \frac{1}{N^2}\mathrm{var}\{g(x)\} \\ &= \frac{1}{N}\mathrm{var}\{g(x)\} \end{aligned} \tag{3.6}$$

This means that as N the number of samples of x increases, the variance of the mean value of G decreases as $1/N$. This result leads to the central idea of Monte Carlo integration that an integral may be estimated by a sum.

$$\begin{aligned} \langle g(x) \rangle &\approx \int_{-\infty}^{\infty} g(x) f(x) dx \\ &= E\left(\frac{1}{N}\sum_{1}^{N} g(x_n)\right) \end{aligned} \tag{3.7}$$

Error in Monte Carlo Estimators

It has been assumed that the variance of a random variable always exists. If it does not exist it is normally better to reform the problem, so that it does exist, otherwise the mean value will converge much more slowly. To show this, the *law of large numbers* is used. This implies that if the random variables $x_1, x_2, x_3, \ldots, x_N$ are independent and all taken from the same distribution such that the expectation of each x is μ, then as $N \to \infty$, the average value of the x's converge to μ

$$P\left\{\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} x_i = \mu\right\} = 1 \tag{3.8}$$

To estimate the error assume that variance exists and use the *Chebychev inequality*. Assuming that an estimator G, its mean $\langle G\rangle$ and variance $\mathrm{var}\{G\}$ all exist, then the Chebychev inequality says

$$P\left\{|G - \langle G\rangle| \geq \left[\frac{\mathrm{var}\{G\}}{\delta}\right]\right\} \leq \delta \tag{3.9}$$

where δ is a positive number. This provides us with the means of estimating the chances of generating a large deviation in the Monte Carlo calculation. Stronger statements about the deviation of an estimate than the Chebychev inequality exist. One such theorem is the *Central Limit Theorem*. This states that for any fixed value N, there is a PDF that describes the values of G that occur in the course of a Monte Carlo calculation. As $N \to \infty$, however, the central limit theorem shows that there is a specific limit distribution for the observed value of G. This distribution is the *normal distribution*

$$\begin{aligned} F(x) &= \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{x} \exp\left|\frac{-(t-\mu)^2}{2\sigma^2}\right| dt \\ F'(x) &= -\frac{1}{\sigma\sqrt{2\pi}} \exp\left|-\frac{(x-\mu)^2}{2\sigma^2}\right| dt \end{aligned} \tag{3.10}$$

Letting $G_N = \frac{1}{N}\sum_n g(x_n)$ the normal form variate is then given by

$$\begin{aligned} t_N &= \frac{(G_N - \langle g\rangle)}{\left[\mathrm{var}\{G_N\}^{\frac{1}{2}}\right]} \\ &= \frac{\sqrt{N}(G_N - \langle g\rangle)}{[\mathrm{var}\{g\}]^{\frac{1}{2}}} \end{aligned} \tag{3.11}$$

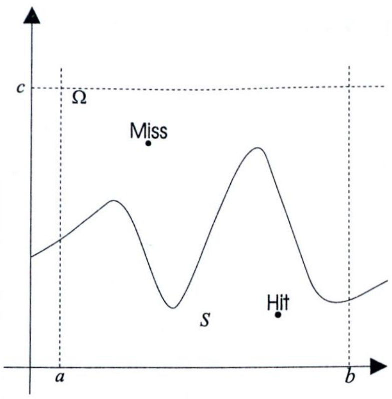


Figure 3.1: Hit or Miss Monte Carlo

then the limiting Cumulative Distribution Function(CDF) is given by

$$\lim_{N\to\infty} P\{a \le t_N \le b\} = \int_a^b \frac{\exp[\frac{-t^2}{2}]dt}{\sqrt{2\pi}} \tag{3.12}$$

By letting $\sigma^2 = \text{var}\{g\}$ in equation 3.11 and substituting t_N for t, this can be written as

$$f(G_N) = \frac{1}{\sqrt{2\pi(\frac{\sigma^2}{N})}} \exp\left[-\frac{N(G_N - \langle g\rangle)^2}{2\sigma^2}\right] \tag{3.13}$$

The central limit theorem is a very good estimator of the probability of deviations measured in units of σ but it only applies asymptotically. We now have the problem of determining how big N has to be before the central limit theorem actually applies. Some answers to this question are given in [136] and [64]. If we cannot determine this we can really only use the weaker Chebychev inequality.

3.1.2 Direct Sampling Monte Carlo Methods

The following are examples of the two most primitive methods used in Monte Carlo integration. Normally they would not be used except as a control to test against other Monte Carlo schemes. However, when combined with one of the many variance reduction schemes, these methods become much more practical.

Hit or Miss Method

This is the most basic of the Monte Carlo Integration methods. To demonstrate this method consider a simple one dimensional integral where we assume that the integrand is bounded.

$$0 \le g(x) \le c$$
$$a \le x \le b$$

Let (X, Y) be a random vector uniformly distributed over the rectangle Ω with PDF

$$F_{XY}(x,y) = \begin{cases} \frac{1}{c(b-a)} & \text{if } (x,y) \in \Omega\ , \\ 0 & \text{otherwise}\ . \end{cases} \tag{3.14}$$

What is the probability p that the random vector (X, Y) falls within the area under the curve $g(x)$? Denoting $S = \{(x, y) : y \le g(x)\}$ and observing that the area under the curve $g(x)$ is

$$\text{area under}\, g(x) = \text{area}\, S = \int_b^a g(x)dx$$

we obtain

$$p = \frac{\text{area}\, S}{\text{area}\, \Omega} = \frac{\int_b^a g(x)dx}{c(b-a)} = \frac{I}{c(b-a)}$$

Let us assume that N independent random vectors $(X_1, Y_1), \ldots, (X_N, Y_N)$ are generated. The parameter p can be estimated by

$$\hat{p} = \frac{N_H}{N}$$

where N_H is the number of occasions on which $g(X_i) \ge Y_i, i = 1, 2, \ldots, N$, i.e. the number of hits and $N - N_H$ is the number of misses. We score a miss if $g(x) < Y_i, i = 1, 2, \ldots, N$ as depicted in Figure 3.1. Thus it follows that to estimate the integral I we take N samples from the distribution, count the number of hits N_H and use the formula below

$$I \approx \theta_1 = c(b-a)\frac{N_H}{N} \tag{3.15}$$

What follows is a sample implementation of this algorithm:

Algorithm

1. Generate a sequence $\{U_j\}_{j=1}^{2N}$ of $2N$ numbers.
2. Arrange the random numbers into N pairs $(U_1, U_1), \ldots, (U_N, U_N)$ in any fashion such that each random number U_j is used exactly once.
3. Compute $X_i = a + U_i(b-a)$ and $g(X_i), i = 1, 2, \ldots, N$.
4. Count the number of cases N_H for which $g(X_i) > cU_i$.

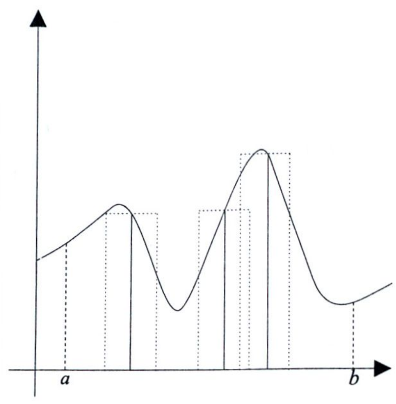


Figure 3.2: Mean Sample Monte Carlo

5. Estimate the integral I by $\theta_1 = c(b-a)\frac{N_H}{N}$.

Mean Sample Method

Another way of computing the integral is to represent it as an expected value of some random variable.

$$I = \int \frac{g(x)}{f_X(x)} f_X(x)dx$$

assuming that $f_X(x)$ is any PDF such that $f_X(x) > 0$ when $g(x) \neq 0$. Then

$$I = E\left[\frac{g(X)}{f_X(X)}\right]$$

where the random variable X is distributed according to $f_X(x)$. If we use equation 3.14 as the PDF then

$$E[g(X)] = \frac{I}{b-a} \text{ and } I = (b-a)E\left[g(X)\right]$$

An unbiased estimator of I is its sample mean

$$I \approx \theta_2 = (b-a)\frac{1}{N}\sum_{i=1}^{N} g(X_i) \tag{3.16}$$

The steps are as follows:

Algorithm

1. Generate a sequence $\{U_j\}_{j=1}^{N}$ of N numbers.

2. Compute $X_i = a + U_i(b-a), i = 1, 2, \ldots, N$.

3. Compute $g(X_i), i = 1, 2, \ldots, N$.

4. Compute the sample mean $\theta_2 = (b-a)\frac{1}{N}\sum_{i=1}^{N} g(X_i)$ which estimates I.

3.1.3 Variance Reduction Techniques

Variance reduction is simply the use of information that is known about a problem and cannot therefore be applied if nothing is known.

Importance Sampling

This method just changes the sampling strategy from using a uniform variate over the region being sampled, to one over the function that is being estimated. Previously it was shown that

$$I = \int \frac{g(x)}{f_X(x)} f_X(x) dx = E\left[\frac{g(X)}{f_X(X)}\right] \tag{3.17}$$

This is an unbiased estimator of I with variance given by

$$\int \frac{g^2(X_i)}{f_X(x)} dx - I^2 \tag{3.18}$$

therefore I is estimated using

$$\theta_3 = \frac{1}{N} \sum_{i=1}^{N} \frac{g(X_i)}{f_X(X_i)} \tag{3.19}$$

The best importance function to use is obviously the function itself. Unfortunately, in order to derive this PDF we need to know the integral of the function. The Markov Chain Monte Carlo methods presented later, provide a means of generating a series of samples from an unknown function. This method is commonly used to sample simple BRDFs such as the cosine lobe [71, 115] and for direct lighting calculations [114, 119, 115].

Stratified sampling

This technique involves partitioning the domain of integration into subregions D_i and evaluating them as separate integrals i.e.

$$I = \sum_{i=1}^{N} \int_{D_i} g(x) f_X(x) dx \tag{3.20}$$

Stratified sampling enables more sampling in the more important regions and also in the smaller regions which might have some known properties and hence can be integrated using another variance reduction technique. This method has not been used much in global illumination problems, but Dutré [34] did use it to partition light sources for sampling.

Control Variates or Extraction of a regular part

This method is based on the principal that given a function $f(x)$ for which an estimate of the integral is required

$$I = \int f(x)dx$$

If there is a function $g(x)$ which approximates $f(x)$ and can be integrated analytically over the domain of integration, then the original integral can be rewritten as

$$\begin{aligned} I &= \int [f(x) - g(x)]\, dx + \int g(x)dx \\ &= \int [f(x) - g(x)]\, dx + C \end{aligned} \tag{3.21}$$

Thus to estimate the new integral

$$\langle I \rangle = \frac{(b-a)}{N} \sum_{i=1}^{N} [f(x_i) + g(x_i)] + C \tag{3.22}$$

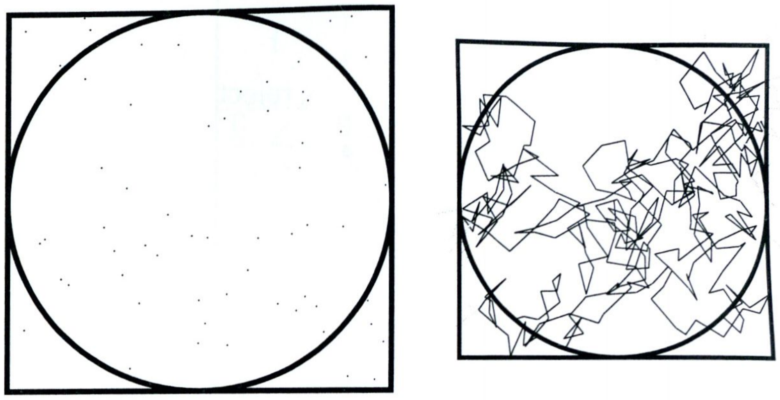
This technique has been implemented in two ways by Lafortune. The first is using a global ambient term [72] and the second is a two pass algorithm using particle tracing to estimate the lighting in sections of the scene [70].

3.1.4 Generation of Random Variables and Samples

In order to be able to sample functions, methods for generating random variates according to a given PDF are needed. To simplify the discussion of these techniques, it is assumed that all samples are taken from a 1-dimensional space. The extension of these techniques to greater dimensionality is straightforward.

Inverse Transform Method Given a CDF $F_X(x)$ for the distribution of the random variate X, the inverse of the CDF must be found. This inverse then gives

$$X = F_X^{-1}(Y) \tag{3.23}$$



(a) Direct Sampling (b) Markov Chain

Figure 3.3: Calculating π

The CDF can be generated from the PDF by integrating it

$$F_X(x) = \int f_X(x)dx \tag{3.24}$$

Thus to obtain a random variate according to $F_X(x)$, generate a uniform random number Y and evaluate $F_X^{-1}(Y)$. The result is a random variate according to $F_X(x)$. This method was used by Shirley and Wang to generate suitable sampling schemes for a variety of shapes [119, 133].

Acceptance-Rejection Method This algorithm is quite simple and allows the generation of a sample according to any PDF $F_X(x)$ and is given by

$$F_X(x) = Ch(x)g(x)$$

where $C \geq 1$, $h(x)$ is also a PDF and $0 \leq g(x) \leq 1$. To generate a random variate according to $F_X(x)$, generate two random variates X and Y from $U(0,1)$ (a uniform random variate in the domain [0,1]) and $h(x)$, respectively. If the inequality $X \leq g(Y)$ holds accept the variable, otherwise try again.

3.1.5 Markov Chain Monte Carlo Methods (MCMC)

A simple example of the use of the naïve Monte Carlo approach is in the calculation of π. To calculate π to an arbitrary number of decimal places, simply generate random numbers in the unit square $[0, 1]$, then calculate how many of these are in a circle inscribed in the square, as shown in Figure 3.3a, this

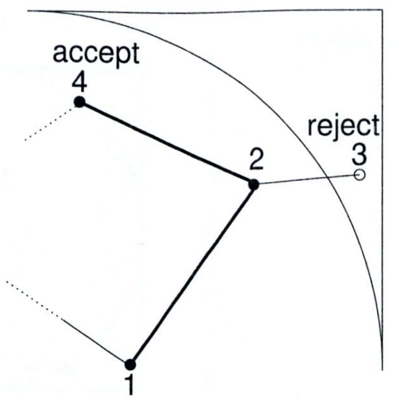


Figure 3.4: Move from Valid sample

is simply hit or miss Monte Carlo. It can be easily verified that the ratio of the number of samples in the square to those in the circle is given by $\pi/4$ (see section 3.1.2).

Direct sampling Monte Carlo Methods such as the method for calculating π just described are very useful but in order to use them effectively some knowledge is required about the domain. Markov Chain Monte Carlo methods are an alternative way of creating samples from a function for which very little information is known. Using our π example again, we can illustrate the differences between the two techniques.

This time random numbers are used to produce a single sample in the square. Following this the random number generator is used to displace the point in a random direction (similar to Brownian Motion, see Figure 3.3b). However, what happens if the point is displaced outside the square? Do we simply keep displacing it until it returns to the square or ignore the displacement and try again? The solution to this problem is the key to the Markov Chain methods and as it turns out neither of the previously mentioned solutions are entirely correct. Instead the current valid sample is re-used and thus displaced again (see Figure 3.4). To see why this works, we'll use the simple puzzle game shown in Figure 3.5. The task here is to create a perfect scrambling algorithm which generates any possible configuration of the puzzle with equal probability. One way to do this is using the naïve algorithm which just breaks up the pieces and places them down in random order. However we are trying to generate a Markov Chain Monte Carlo (MCMC) alternative. We could just randomly move the blank square in a random direction but this does not produce all configurations with equal probability, due to the limitation of choices at the boundaries as shown in Figure 3.6. If the algorithm generates the configurations a, b and c with equal probabilities $\xi(a)$, $\xi(b)$ and $\xi(c)$, respectively, we can then derive a

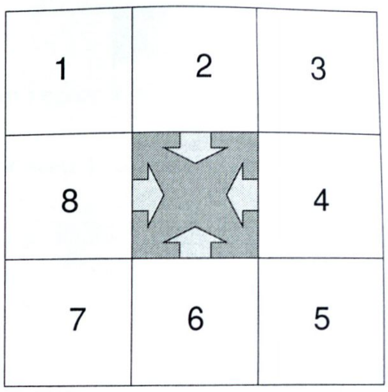


Figure 3.5: Sliding Puzzle

simple rate equation relating the ξ's to the transition probabilities $p(x \to y)$ where x and y replace the various configurations of a, b and c. Given that $\xi(a)$ can only be generated from b, c or itself we get

$$\xi(a) = \xi(a)p(a \to a) + \xi(b)p(b \to a) + \xi(c)p(c \to a) \tag{3.25}$$

which gives

$$\xi(a)[1 - p(a \to a)] = \xi(b)p(b \to a) + \xi(c)p(c \to a) \tag{3.26}$$

If we impose the condition that tells us that the empty square, once at a, can stay at a or move to b or c.

$$1 = p(a \to a) + p(a \to b) + p(c \to a) \tag{3.27}$$

then substituting this into equation 3.25 gives:

$$\xi(a)p(a \to b) + \xi(a)p(a \to c) = \xi(c)p(c \to a) + \xi(b)p(b \to a) \tag{3.28}$$

We can satisfy this equation by equating the braced terms separately giving the *detailed balance condition*

$$\xi(a)p(a \to b) = \xi(b)p(b \to a) \tag{3.29}$$
$$\xi(a)p(a \to c) = \xi(c)p(c \to a) \tag{3.30}$$

From this it can be seen that the simplest way of maintaining the probability is to propose an equal probability of $1/4$ for any possible move, thus at the corner we have to be immobile (reject the move) with probability $1/2$ and with $1/4$ on the edges. So now looking back to the π estimation problem we see that

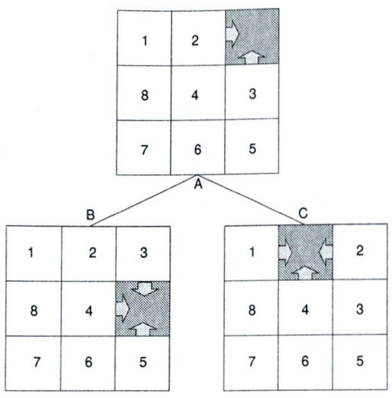


Figure 3.6: Puzzle Steps

anytime we cross the boundary we must remain immobile to maintain detailed balance (see Figure 3.4).

Metropolis Sampling

Metropolis sampling is a form of importance sampling which produces an importance function which tries to mimic the behaviour of the function being sampled. However, it only starts to do this after a certain number of samples are taken. The selection of samples is viewed as a Markov process in an equilibrium system with a distribution function $f(x)$. In this equilibrium, the values of the distribution function at different points are related by the *detailed balance* condition described previously:

$$f(x)A(x \to x')T(x \to x') = f(x')T(x' \to x)A(x' \to x), \tag{3.31}$$

where $T(x \to x')$ is the transition rate from the state x to the state x' and $A(x \to x')$ expresses the probability of attempting a move from x to x', called the proposal distribution. A and T together form the transition probability $p(x \to y)$ from equation 3.30. Thus the samples are no longer taken randomly but by following a Markov Chain according to $f(x)$. These samples are accepted if the ratio of the transition satisfies

$$\frac{T(x \to x')}{T(x' \to x)} = \frac{f(x')A(x' \to x)}{f(x)A(x \to x')} \geq \zeta_i, \tag{3.32}$$

where ζ_i is a uniform random number in the region $[0, 1]$. This results in

$$T(x \to x') = min\left(1, \frac{f(x')A(x' \to x)}{A(x \to x')f(x)}\right). \tag{3.33}$$

The method works as follows:

Algorithm

```
1. generate a random vector $x_i$ in the function domain $\Omega$
2. if $f(x_i) = 0$ repeat step 1, otherwise continue
3. mutate $x_i$ producing $x_{i+1}$ by random displacement
4. generate random number $\zeta_i$
5. if $\frac{f(x_{i+1})A(x_{i+1}\rightarrow x_i)}{A(x_i\rightarrow x_{i+1})f(x_i)} \geq \zeta_i$ accept $x_{i+1}$ otherwise use $x_{i+1} = x_i$
6. repeat steps 3 to 5
```

Usually for most situations $A(x_i \rightarrow x_{i+1})$ is symmetric and hence can be removed from the equation. What we have described is the Metropolis-Hastings algorithm. There are some far more general forms of this kind of sampling that use measures rather than probabilities. This is beyond the scope of this review, for more information see [36].

3.1.6 Re-parameterisation

Another way of increasing the performance of Monte Carlo techniques is to re-parameterise the functions you are using. Most of the best re-parameterisations usually reduce the dimension and thus the effect of n-dimensional explosion. This in turn increases the convergence of the technique. Direct sampling Monte Carlo with random sampling, does not seem to suffer badly from the n-dimensional explosion and hence a reduction in dimension does not produce as great an increase in efficiency. Nevertheless, other parameterisations are possible. These tend to be very specific to the problem domain. The following sections describe some common re-parameterisations used in global illumination.

Eye Ray Tracing

This is where the light path is parameterised via a path starting at the eye which eventually reaches a light source. Approaches which use this parameterisation are usually called gathering methods since they seem to gather all light coming from the light source. This parameterisation is very good at solving for diffuse illumination such as shadows and colour bleeding but light focusing effects such as caustics are generally poorly estimated, usually due to the way the final ray to the light source is parameterised. This is the scheme used in most path tracing schemes such with Kajiyas method [63].

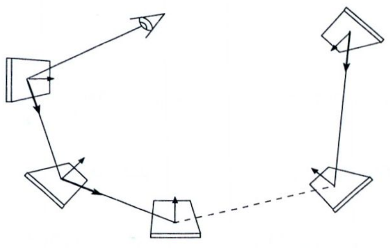


Figure 3.7: Bi-directional Path Tracing

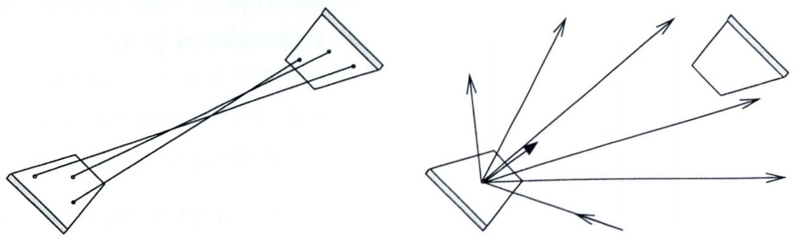
Light Ray Tracing

Using this a ray path is parameterised from the light source to the eye. This is basically the opposite of the previous method. Methods which use this parameterisation are called shooting methods, since they appear to be firing light into the scene. This method excels at solving most lighting conditions but when used to construct a path to areas which are difficult to hit (such as the film plane of a camera) it produces poor estimates. This is because the probability of successfully negotiating very small targets such as the camera aperture is very low. A number of different algorithms have used the light ray tracing parameterisation. The first of these was probably used in Arvo's backward ray tracing [6], nearly all particle tracing techniques use this parameterisation [52, 62, 22, 131].

Bi-Directional Path Tracing

Bi-directional path tracing (see Figure 3.7) is an algorithm which takes into account the importance of both the viewing point and the light sources present in the scene. It does this by starting ray paths at both the eye and the light sources. These paths are then joined together and the light that travels along the resulting path computed. By varying the ratio of bounces between the eye and light paths, different lighting situations can be more accurately estimated. This technique can be used to construct single paths joining points on the film to light sources. There are two slightly different formulations for the use of this parameterisation, Veach's method [129] and Lafortunes [69]. Their differences principally lies in how the results are combined and also their use of shadow rays.

Some other parameterisations that can be be applied are the surface to surface (the ray is parameterised by its start and end points on two surfaces, using their parametric surface coordinates), and solid angle parameterisations (the ray is parameterised by a position and a direction) of these equations (see Figure 3.8). Normally the best parameterisation to use is the solid angle formulation since it does not have any singularities (situations where the result goes to infinity, this



(a) Surface to Surface (b) Solid Angle

Figure 3.8: Ray parameterisations

happens in the surface to surface variation when the d term goes to 0) which the surface to surface formulation does. When doing direct lighting or radiosity-like methods, the surface to surface parameterisation proves to be more practical despite its limitation.

3.2 Pseudo Random Variates

Choosing N points uniformly randomly in an n-dimensional space leads to an error term in Monte Carlo integration that decreases asymptotically as $N^{-\frac{1}{2}}$. Thus each new point sampled adds linearly to an accumulated sum that will become the function estimate. If points are chosen, say on a Cartesian grid, then the Monte Carlo method becomes a deterministic quadrature scheme whose fractional error decreases asymptotically fast as N^{-1}. Unfortunately, using a Cartesian grid limits the number of sample points. To enable adaptive Monte Carlo techniques where the number of samples is unknown, quasi-random sequences such as the *Halton* sequence can be used. This sequence is described in Section 3.2.1. These have been shown to reduce the variance in a similar manner. Quasi random variates have been used by Keller [65, 54, 66] with Radiosity based methods of solution and also on the viewing plane [53].

3.2.1 Quasi Random Sampling

Quasi random sampling deals with the use of a deterministic rather than a random set of samples. The idea is to distribute as evenly as possible the samples through out the parameter space. As a side effect of the samples being deterministic, error assessment cannot be based on probabilistic reasoning. Instead to compare quasi random sequences the idea of *discrepancy* is used. This is basically used to measure the effectiveness of a sequence. To compare quasi random sequences there are various discrepancy measures. When approximating the distribution function of the uniform distribution on $[0,1]^k$, the *discrepancy* D_n is defined as the supremum error over all sub-rectangles of $[0,1]^k$.

Van Der Corput

Let $r < 1$ be an integer and also that $m \in \mathbf{N}$ can be uniquely expressed in base r as $m = a_o + a_1 r + a_2 r^2 + \ldots + a_l r^{l-1}$ where each $a_i \in \{0, 1, \ldots, r-1\}$, $a_l \neq 0$ and $r^l \leq m < r^{l+1}$. The base r *radical inverse* function is given by

$$\phi_r(m) = a_0 r^{-1} + a_1 r^{-2} + a_2 r^{-3} + \ldots + a_l r^{-l-1} \tag{3.34}$$

The radical inverse function thus reflects the numbers through the decimal point. The sequence $x_i = \phi_r(i)$ is called the *Van der Corput* sequence. It can be shown that $D_n^* = O(n^{-1} \log n)$, confirming that it is indeed an equidistributed sequence.

Halton sequences

The Van der Corput sequence is a quasi random sequence for one dimension. To extend this to n dimensions, the *Halton* sequence uses a different base or radix for each dimension. These bases must satisfy certain criteria so as to avoid correlations. To achieve this, k bases $r_1, r_2, r_3, \ldots, r_k$ are chosen such that they are pairwise relatively prime. It is common to take the bases as being the first k primes. The i-th element of the Halton sequence is given by

$$x_i = (\phi_{r_1}(i), \ldots, \phi_{r_k}(i)). \tag{3.35}$$

It can be shown that

$$D_n^* \leq \left[\sum_{i=1}^{k} \frac{n-1}{2\log r_i}\right] \frac{(\log n)^k}{n} + \frac{(\log n)^{k-1}}{n} \tag{3.36}$$

This implies that $D_n^* = O(n^{-1}(\log n)^k)$ but now there is a dimensional effect implicit in the asymptotic error rate for convergence. If we take the asymptotic error rate for Monte Carlo to be $n^{-1/2}$ then

$$n^{-1}(\log n)^k / n^{-1/2} = n^{-1/2}(\log n)^k \to 0 \text{ as } n \to \infty \tag{3.37}$$

Thus the integration using the Halton sequence has a better asymptotic error rate for all dimensions but the Monte Carlo method does not have an explicit dependence on the dimension.

Hammersly sets

The Hammersly set is a simple extension of the Halton sequence whereby the first dimensions sequence is given by i/n instead of using a Van der Corput sequence. Thus, it is given by

$$H_n = \left\{ \left(\frac{i}{n}, \phi_{r_1}(i), \ldots, \phi_{r_{k-1}}(i) \right) : i = 0, \ldots, n-1 \right\} \tag{3.38}$$

It can be shown for the Hammersly set that

$$D_n^* \leq \left[\sum_{i=1}^{k-1} \frac{n-1}{2 \log r_i} \right] \frac{(\log n)^{k-1}}{n} + \frac{(\log n)^{k-2}}{n} \tag{3.39}$$

which gives that $D_n^* = O(n^{-1}(\log n)^{k-1})$, which is better than that of the Halton sequence. However, to increase the number of points in the Hammersly set we must recompute the set, but if we use the Halton sequence we can simply reuse the points.

3.3 Solution of Integral Equations

So far methods of solving integrals using Monte Carlo Quadrature have been discussed. How are these methods used to solve integral equations? The following methods of approximating integral equations must obey the constraints that $b(x)$, e and $K(x, y)$ (see Appendix B) must be Lesbesgue integrable i.e.

$$\int b^2(x)dx \leq \infty$$
$$\int e^2(x)dx \leq \infty$$
$$\int\int K^2(x, y)dxdy \leq \infty$$

and that

$$|K| \leq 1$$

3.3.1 Using Recursion

Given an integral operator K, a solution is to recurse through the equation evaluating each term in succession, using any integration method suitable for evaluating that integral.

$$\begin{aligned} b &= e + Kb \\ &= e + K(e + Kb) \\ &= e + K(e + K(e + Kb)) \end{aligned} \tag{3.40}$$

This is the method used in distributed ray tracing [24]. It is analogous to gathering all the light incoming to a surface point. Unfortunately, this method is very slow as it puts too much effort into solving rays which do not contribute much to the solution. Despite its slowness, it does produce very good looking images.

3.3.2 Inversion

The solution can also be obtained by inverting the operator $I - K$.

$$b = (I - K)^{-1}e \tag{3.41}$$

Usually it is impractical to invert the integral operator except for very simple cases. For more complex cases such as global illumination problems, K needs to be mapped to a finite dimensional domain as is done in Finite Element Methods (FEM). This is essentially the method used by matrix based radiosity approaches [86, 45]. These methods are not covered in this dissertation but for a comprehensive review with respect to image synthesis see [19].

3.3.3 Newman Series Approximation

Approximate solutions to many integral equations can be found iteratively. Starting with some initial guess $b^{(0)}(0) = e$, subsequent approximations are defined by

$$b^{(i)} = e + Kb^{(i-1)} \tag{3.42}$$

If we start with $b^{(0)} = e$, then the i^{th} approximation is the truncated series

$$b^{(i)} = e + Ke + K^2e + \ldots + K^ie$$

where K^i denotes i successive applications of the integral operator K. The sequence $b^{(i)}$ converges if the norm of the integral operator is less than 1

$$||K|| < 1$$

where the operator norm is defined in terms of a function norm

$$||K|| = max_{b\neq 0}\left(\frac{||Kb||}{||b||}\right)$$

The L_p norms are a general class of function norms

$$||f||_p = \left(\int_\beta^\alpha f(x)\hat{}-p''dx\right)^{1/p}$$

the most common of which are the L_1, L_2 and the L_∞ norms. The exact solution to the integral equation is given by the Neumann series

$$b = b^{(\infty)} = \sum_\beta^\alpha K^i e$$

If $||K|| \geq 1$, the Neumann series diverges. The Neumann series is a generalisation of the geometric series for $1/(1-a)$. Not many global illumination techniques have made use of this method, however, Szirmay-Kalos' stochastic iteration [123] is one such method which has.

3.4 Expansion

Expansion works by expanding out the integral equation as follows

$$\begin{aligned} b &= e + Kb \\ &= e + K(e + Kb) \\ &= e + Ke + K^2(e + Kb) \end{aligned} \tag{3.43}$$

Since this is essentially an infinite expansion, we end up with the following Neumann series

$$b = \sum_{i=0}^{n} K^i e + K^{n+1} b \tag{3.44}$$

If K is a contraction (i.e. $lim_{n->\infty} K^{n+1}b = 0$) then we are left with

$$b = \sum_{i=0}^{n} K^i e \tag{3.45}$$

Where the first term is the direct emission, the second is the light from a single scatter and so on. The expansion method therefore describes the random walk methods such as path tracing [63] and bi-directional path tracing [129, 69].

3.5 Density Estimation

Density estimation deals with the construction of an estimate of a density function, using a set of sample points that have been drawn from an unknown probability density function. In our case, this is the estimate of the flux across a surface given a set of photons (the samples), which are generated from the potential equation (the probability density function). There are two approaches to density estimation: *parametric*, where the data is drawn from a known parametric family of distributions such as the normal distribution; and the *non-parametric* where no such assumptions are made. In our discussion we are only interested in nonparametric density estimation because no parametric assumptions can be made about the samples that we are aware of.

3.5.1 Methods for density estimation

What follows is an overview of various density estimation methods, this is adapted from that given in Silverman [121].

Histogram

The histogram is perhaps the most obvious density estimation method. It basically entails breaking the domain up into bins and counting the number of samples which fall into those bins. This is given by:

$$f(x) = \frac{1}{nh}(\text{no. of samples in same bin as x}) \tag{3.46}$$

where n represents the total number of samples and h is a measure of the size of the bin corresponding to the dimension used (e.g. length in 1-D, area in 2-D and volume in 3-D). Unfortunately, the histogram offers no smoothing and the results are particularly sensitive to the start position of the bins and also to the bin size. This can cause misleading results due to aliasing. However, these methods are very fast. The histogram has been used in a number of particle based methods such as Heckbert's radiosity textures [52]. Collins' splating technique [22] adds smoothing to illumination maps, using a hybrid of histograms and the kernel estimator method mentioned later.

Naïve Estimator

The naïve estimator is a variant on the histogram method. The bins are no longer set at fixed positions but instead centred around the point at which we are trying to estimate the density, known as the *observation*. This is given by the following

$$f(x) = \frac{1}{2hn}(\text{no. of samples in } (x-h, x+h]) \tag{3.47}$$

where h and n are the same as in the histogram method. This produces a better result than the histogram but unfortunately is also not continuous.

Kernel Estimator

The kernel estimator is a more general form of the naïve estimator, where instead of treating all samples the same, a kernel function K is used to weight their contribution. The kernel is constrained such that

$$\int K(x)dx = 1 \tag{3.48}$$

Normally the kernel is symmetric but this is not always the case. The kernel estimator is given by the following

$$f(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right) \tag{3.49}$$

where n, h and x are as in the previous methods and X_i represents a sample that was found inside the observation window. Sometimes with these types of estimators h is called the *window width*, *bandwidth* or *smoothing parameter*. Just as the naïve estimator is considered as a sum of boxes, the kernel estimator is considered a sum of bumps. These methods produce continuous density estimates, but they cannot deal effectively with the boundaries of a domain. This is because there are no samples beyond the boundary and therefore bias is introduced.

Nearest Neighbour

This method is based on finding the k nearest neighbours to the observation position and using them to estimate the density

$$f(t) = \frac{1}{nd_k(t)}\sum_{i=1}^{n} K(\frac{t-X_i}{d_k(t)}) \tag{3.50}$$

where $d_k(t)$ is the distance to the k nearest sample from the observation point t. Thus the overall smoothing is governed by the choice of k. This technique is used in Jensen's photon map [61].

Kernel	$K(u)$
Uniform	$\frac{1}{2}\mathbf{I}(\|u\| \leq 1)$
Triangle	$(1-\|u\|)\mathbf{I}(\|u\| \leq 1)$
Epanechnikov	$\frac{3}{4}(1-u^2)\mathbf{I}(\|u\| \leq 1)$
Quartic	$\frac{15}{16}(1-u^2)^2\mathbf{I}(\|u\| \leq 1)$
Triweight	$\frac{35}{32}(1-u^2)^3\mathbf{I}(\|u\| \leq 1)$
Gaussian	$\frac{1}{\sqrt{2\pi}} \exp -\frac{1}{2}u^2$
Cosinus	$\frac{\pi}{4}\cos\left(\frac{\pi}{2}u\right)\mathbf{I}(\|u\| \leq 1)$

Table 3.1: Kernel Functions

Variable Kernel

This approach is similar to the previous method but this time the smoothing is adapted to the local density of the data. This is given by

$$f(t) = \frac{1}{n}\sum_{j=1}^{n} \frac{1}{hd_{j,k}} K\left(\frac{t - X_j}{hd_{j,k}}\right) \tag{3.51}$$

where h is the smoothing parameter, $d_{j,k}$ is the distance from X_j to the k^{th} nearest point and t is the point at which we wish to estimate the density. Thus the samples in areas where there are relatively few samples will have flatter or wider kernels than those in more densely populated areas, which will have smaller kernels.

There are many other methods for determining a density estimate but we have chosen to focus on the above methods due to their simplicity and ease of implementation. For information about other methods see Silverman [121]. Bruce Walter also has an alternative density estimation method, based on regression which gives a meaningful way to estimate the density at the boundaries of surfaces [131].

3.5.2 Kernel Types

Up until this point kernels have been described but no details given about them. As already mentioned, a kernel is a function which expresses the contribution of a sample to a domain. There are many choices of function for a kernel, but according to Silvermann no one kernel appears to give sufficiently better accuracy than any other, to make it the obvious choice. To develop this he suggests that the choice of kernel should be guided by the type of function that is being estimated. Table 3.1 and Figure 3.9 shows some sample kernel functions, the most popular of these tend to be the Epanechnikov or the Gaussian. Note that in the table $\mathbf{I}(\rho)$ represents the indicator of a relation, returning 1 if the relation ρ is satisfied and 0 otherwise.

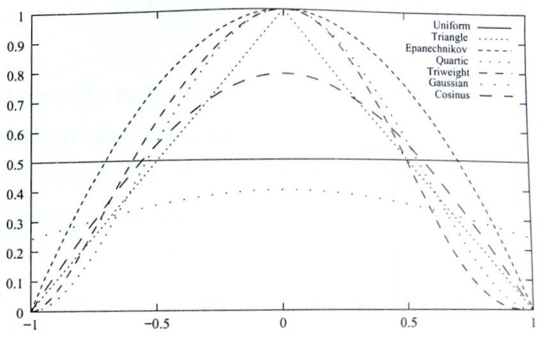


Figure 3.9: Kernel Shapes

Error and Bias

The problem now is with the reconstruction of the function. What is the best bandwidth to use? The most natural way of determining the error is to use the *mean squared error*. This is given by

$$\begin{aligned} MSE\langle \hat{f} \rangle &= E\{\hat{f}(x) - f(x)\}^2 \\ &= (E\{\hat{f}(x)\}^2 - f(x))^2 + var\{\hat{f}(x)\} \end{aligned} \tag{3.52}$$

where $\hat{f}$ is the estimation of the function. This is the sum of the squared bias and the variance at x. Unfortunately there is a trade-off between bias and variance. By reducing the bias we increase the variance and vice versa. This bias is basically controlled by changing the bandwidth. We note that there are many other ways of measuring the error but this method is similar to the method used with integration schemes previously discussed and is therefore familiar and widely used in the graphics community. It does however leave out any perceptual properties, which are addressed directly by Walter [132]. To find the total error, the *mean integrated square error* is used, given by

$$\begin{aligned} MISE\langle \hat{f} \rangle &= \int E\{\hat{f}(x) - f(x)\}^2 \\ &= \int (E\{\hat{f}(x)\}^2 - f(x))^2 + \int var\{\hat{f}(x)\} \end{aligned} \tag{3.53}$$

where the MISE is the sum of the *mean integrated square bias* and the total variance. Using this formula Silvermann shows that the bias does not depend directly on the sample size but on the bandwidth and to a lesser extent on the kernel. This formula can be used to determine good parameters for the density estimation and the choice of kernel is dependent on the application. According to Collins [21] the Gaussian kernel appears to be the best choice. Consequently we have also used this as our primary kernel choice.

3.6 Summary

The Monte Carlo techniques can be used in a number of ways to estimate values and sample functions used by our models. Monte Carlo techniques are useful in a wide range of applications, not just as methods of integrating equations. They can be used to sample functions and to estimate errors and their convergence rates can be determined. The main benefit of these techniques are the fact that in most cases their convergence criteria are not implicitly affected by the dimensionality of the problem, as almost all other numerical techniques are. For instance, the Simpson's rule integration scheme has more accuracy per sample but only in one or two dimensions. Higher dimensions require n^m samples which explodes as the dimensions increase, whereas with Monte Carlo schemes each sample is generally independent of each other. Thus any variety of samples can be used to get an estimate. The benefits of the Monte Carlo methods do not stop there: Not only can they be used to estimate integrals but also to sample complex functions. These samples can then be used in a number of ways. The illumination across a surface can be estimated by using density estimation techniques, which have the ability to remove the noise caused by random sampling at the cost of introducing bias, or the samples can be used to improve an integration scheme via importance sampling. The noise produced by the random sampling process and the $N^{-\frac{1}{2}}$ asymptotic convergence rate, are a thorn in the side of Monte Carlo methods especially where image synthesis is concerned, where a good estimate is required from a relatively small number of samples. On the other hand, by using various techniques these side-effects can be reduced usually at the cost of introducing correlations in the image or increasing the bias.

Chapter 4

Parallel Processing

Even for simple scenes global illumination calculations take a relatively long time to generate, compared to non-realistic rendering methods. This is not because the techniques involved are that different, but because more computations such as ray casting are necessary in order to achieve more realistic images. Another difference is that the amount of data used and created during the rendering process is correspondingly higher, thus causing problems due to the limited memory on standard computers. To reduce the time taken to create images and to address the memory consumption of these algorithms, many parallel methods have been devised. By using more than one computer, the computational load as well as the the memory load may be distributed. This speeds up rendering times and increases the amount of data that can be used and created during rendering. Unfortunately, parallel rendering introduces its own problems which reduce its effectiveness. This means that the ideal speed and memory increase of n times for n machines can probably never be achieved, except in very simple cases. This chapter introduces some of the concepts, pitfalls and solutions used with parallel processing in relation to global illumination problems. A more detailed exploration of these issues may be found in [16].

4.1 Concepts

Parallel processing is the division of a task into a number of subtasks which can then be solved by separate processes. This usually involves some communication, even if only to initialise the distribution of work. A simple analogy is to imagine planting potatoes: If two people perform the task it should be completed twice as fast as using one person. However, as more and more people start digging, problems start occurring due to limited resources, such as the number of forks available. Eventually, there will not be enough room to dig.

Similar problems are experienced when creating parallel programs.

Task dependencies are also a problem, as certain tasks are restricted by the order in which they can be performed. For example, planting the potatoes or creating the drills can only be performed after the field has been dug. *Pipelining* is one method used to combat the negative effects of dependencies. This method works well when a task consisting of a number of distinct stages needs to be executed repeatedly. Going back to our potato planting example, we can demonstrate the use of pipelining, the following being the dependent tasks:

1. dig patch,
2. create drill,
3. add manure,
4. add fertiliser,
5. plant potato,
6. cover potato with soil.

This is essentially a pipeline consisting of 6 stages. If we use 6 people we can start one person off digging. When they have finished a patch the next person starts creating a drill. After the initial part of the drill is created the next person adds the manure and so on until finally the last person creates the drill over the potato. This sets up a work pipeline, whereby each person starts after the other has performed his first task. The plan has a flaw, in that some tasks such as digging require more time than planting, but no task can be faster than any task that is ahead of it. Thus the pipeline is only as fast as the slowest task.

Another problem with parallel tasks is that they don't necessarily scale well. Adding more processors will therefore not necessarily decrease the time taken to complete a job. This is known as the *scalability* of a task. Finally, all parallel tasks need some form of control to tell them what work to perform. There are two ways of solving this problem: One is to have a *centralised* controller or master, whereby one worker is in charge of all the the others. The alternative is to have *distributed* control, by allowing each worker to organise their own access to resources. For instance, in our example, allowing the first person that gets a spade to use it would be an example of distributed control, whereas having a foreman tell each worker what to do would be centralised control.

4.2 Classification

A classical sequential computer is based on the *von Neumann* architecture. This architecture consists of a processor, memory, an I/O interface and various buses

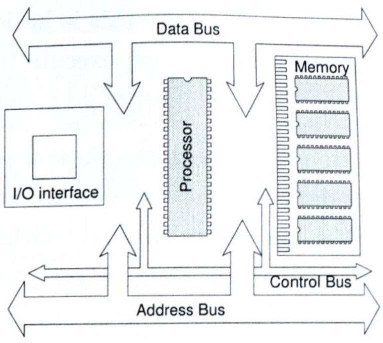


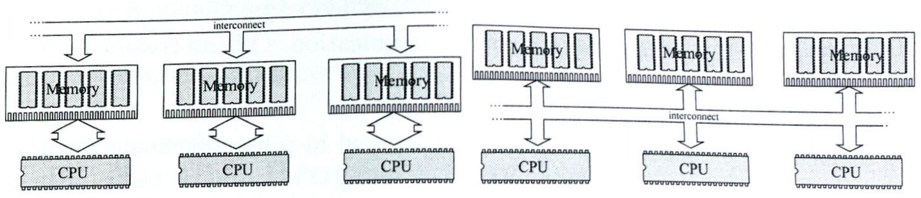
Figure 4.1: Von Neumann Architecture

connecting these units (see Figure 4.1). The processor is the unit responsible for fetching, decoding and executing instructions, which are retrieved from RAM. Most of today's computers conform more or less to this specification. Over the last few years many techniques from parallel processing have been exploited to increase performance. Pipelining was added to the fetch, execute and decode cycle, as seen in Intel's 80486 and Motorola's 68020. Parallel execution has been added in the form of multiple execution units, such as integer or floating point units to Intel's Pentium and Motorola's 68060. *Vector processing*, whereby a single instruction is performed simultaneously on a vector of data (usually held in a register), is another form of parallel execution. This has been added to the von Neumann architecture in Intel's Pentium MMX [95, 13] series and the Motorola PowerPC as AltiVec [30]. This feature provides great performance gains, where large arrays of similar data need to be processed. Lastly, multiple *processing elements* (PE's) are becoming more common. This is known as *Symmetric Multi-Processing*. This is beginning to emerge as an alternative way to improve the performance of desktop computers by allowing tasks to be truly run in parallel, not just *concurrently* via time sharing.

4.2.1 Flynn's Taxonomy

There have been many different parallel architectures proposed, many of which have been realised. This has led to the desire to create a classification for these systems. One of the most widely known, if not used, classification schemes is Flynn's Taxonomy [37], despite being criticised heavily for being too simple. However, because of its simplicity and the fact that various details concerning parallel architectures are not really an overwhelming concern to many programmers, it is widely used. The classification basically breaks parallel architectures into four main categories:

- SISD Single Instruction Single Data - this is basically the von Neumann architecture as found in all PCs which execute instructions sequentially on a single data stream.
- SIMD Single Instruction Multiple Data - these machines apply a single instruction to multiple data streams, for instance the Intel's MMX [95, 13] and Motorola's AltiVec [30] instruction set. Other notable examples include array processors such as the Connection Machine [57], the Goodyear Massively Parallel Processor (MPP), also vector processors such as the NEC SX1 and the Fujitsu FACOM VP-200.
- MISD Multiple Instruction Single Data - although no architecture falls into this category, some architectures seem similar to this concept such as systolic arrays.
- MIMD Multiple Instruction Multiple Data - this is essentially a group of processing elements which apply their own instructions to their own data stream. The SMP machines or a network of PCs would be in this group. These systems are classified as *tightly coupled* if the degree of interaction between processors is high, or *loosely coupled* if it is low. There are three methods of achieving this goal (see Figure 4.2): One is *shared memory* as is found in many SMP configurations where all processors share a common address space. The second is a *distributed memory system* where each processor has its own private memory and communicates by passing messages via a communication path such as a network of PCs. The third is a hybrid of the two previous types, called a *distributed shared memory system.* For large numbers of processors, shared memory systems are not feasible since there are too many contention problems. Distributed systems present a similar problem, but by creating dedicated links between processors the contention issues can be reduced, thus making such an approach more feasible. To distinguish between a cluster of workstations and sets of interconnected dedicated processors with memory nodes, the term *parallel system* is used to refer to the dedicated system. *Distributed system* is used to refer to the cluster of work stations. The distinguishing factors between these two systems are the computation-to-communication ratio and the cost, the cluster of workstations having a higher communication overhead and being cheaper to assemble. Examples of MIMD computers include SGI's Origin series, which is a distributed shared memory system. It has both local memory and shared memory which is used to communicate with other processors. The IBM SP/2 is a loosely coupled system which uses a custom designed high speed network along which PEs communicate.



(a) Distributed Memory (b) Shared Memeory

Figure 4.2: Shared Memory Systems

The most powerful computer at the time of writing is ASCI Red, which is a distributed memory, MIMD, message-passing supercomputer. Currently it consists of 9536 Pentium II Xeons in its core system, which gives it a peak performance of 3.15 Terra-flops. This system was designed to be highly scaleable with respect to both the number of processing units and also the communication bandwidth. The current arrangement is not yet at the limits of its design, according to its developers.

4.3 Tools for MIMD Systems

A number of tools have been created to aid the programmer in creating portable parallel applications. The two main tools used are the Parallel Virtual Machine [10] (PVM) and the Message Passing Interface [3] (MPI). These tools provide a set of common functions which can be implemented on a variety of MIMD architectures, such that the programmer need not worry about the subtleties of the system they are using, while at the same time providing an acceptable level of optimisation.

4.3.1 Message Passing Interface (MPI)

The MPI library is an Aplication Programming Interface (API) based on the message passing paradigm, which is used widely on certain classes of parallel machines, especially those with a distributed memory architecture. MPI is a standard which attempts to make use of the most attractive features of a number of existing message passing systems, rather than selecting one of them and adopting it as the standard. The goals of the MPI standard are as follows (from [1]):

- Design an API (not necessarily for compilers or a system implementation library).

- Allow efficient communication: Avoid memory-to-memory copying and allow overlap of computation and communication. Offload communication to a coprocessor, where available.
- Allow for implementations that can be used in a heterogeneous environment.
- Allow convenient C and Fortran 77 bindings for the interface.
- Assume a reliable communication interface: the user need not cope with communication failures. Such failures are dealt with by the underlying communication subsystem.
- Define an interface that is not too different from current practice, such as PVM, NX, Express, p4, etc., and provides extensions that allow greater flexibility.
- Define an interface that can be implemented on many vendor's platforms, with no significant changes in the underlying communication and system software.
- Semantics of the interface should be language independent.
- The interface should be designed to allow for thread-safety.

MPI was used for all the parallel algorithms presented in this thesis.

4.3.2 Parallel Virtual Machine (PVM)

PVM, as its name implies, is different from MPI in that instead of being an API, it tries to form a virtual parallel machine. It is not a standard as such, but a system developed by researchers at Oak Ridge National Laboratory, and may be used to create parallel solutions to a given problem. For further information concerning the differences between MPI and PVM refer to [39]. PVM is based upon the following principles (from [2]):

- User configured host pool - the setup of the parallel machine is configured by a user who selects the machines they wish to use.
- Transparent access to hardware - the machines making up the PVM may be seen as a set of generic processing elements, or instead the specific capabilities of each machine may be queried and used.
- Computation is process based - the unit of parallelism being the task.
- Explicit message passing model - tasks cooperate by passing messages between each other.

- The ability to support a heterogeneous environment.
- Multiprocessor support.

4.4 Parallel Programming Issues

A parallel program consists of a number of subtasks, each of which may run on a separate processor, using some form of communication to coordinate their efforts. The coordination effort usually introduces some new types of program bugs that mainly affect parallel programs. With serial programs it is easy to determine the location of a bug, because the bugs are directly related to the program itself, whereas in a parallel program they can be introduced by the coordination effort. This situation is especially difficult to debug when large numbers of processors are involved. A common bug that occurs is known as *deadlock*, which occurs when a processor is made to wait indefinitely for an event that will never occur. Contention over a resource such as memory is a common cause of deadlock (see Figure 4.3). Another problem, perhaps more difficult to detect and fix, is that of *data consistency*. It is caused by the fact that data is distributed among a number of processors which may modify the data, thus causing differences between the data on each processor. A number of methods are used to solve these problems, such as:

- keeping the data constant,
- dividing the data amongst the processors such that there is no duplication,
- fetching and storing data to and from one location.

Serial algorithms can devote one hundred percent of their processing time to the problem. Parallel implementations on the other hand must dedicate a percentage of their time to communicating with other processors. For this reason, some processors may become idle while they wait for data or tasks. In order to produce an effective parallel algorithm, this *computation to communication ratio* must be maximised.

4.4.1 Tasks

The process of dividing a problem amongst many processors creates a number of tasks which need to be performed. *Tasks* are simply the units of computation assigned to each processor. The *task granularity* is the amount of computational effort required to complete a task, and is directly related to the computation to communication ratio. The more time required to complete a task, the less suited

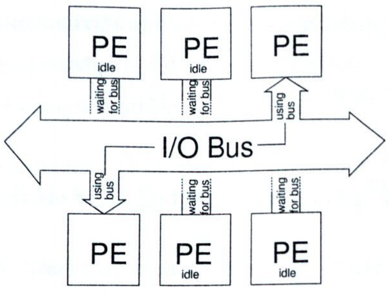


Figure 4.3: Deadlock as other PE's wait for I/O Bus

it may be to a parallel system, while a smaller granularity will cause more time to be devoted to scheduling issues because tasks are completed more rapidly.

4.4.2 Data

Many of the problems that can be solved by parallel methods create and use large amounts of data, which may not fit in the memory of a single processor. One way of using parallel processors to help solve this problem is to divide the data amongst the processors such that the total memory available is the sum of the memory available to each processor. However, now the communication overheads may become bigger as tasks must be migrated to where the data is or vice versa. This causes idle time while data or tasks are being transfered. It is the job of the scheduling algorithm to reduce this delay to a minimum, thus making the parallel algorithm more efficient.

4.5 Evaluation of Parallel Implementations

The only real reason for using a parallel implementation in global illumination is to speed up the solution of a problem. Therefore time taken to complete a parallel algorithm is a good measure for determining the benefits of this approach. Other measures such as efficiency and speed-up give insights into the scalability of an algorithm. Comparing parallel implementations can be difficult, especially if completely different hardware is used in each case. The optimisations present in a serial program may not always be usable in a parallel scenario, thus making comparisons more difficult. A common misconception is that adding more processors to a solution will always obtain the result faster, which is not always the case. The *realisation penalty* is the effect caused by the communication overheads of an algorithm. As you add more processors, progressively more time is wasted communicating, until eventually the communication overheads actually start to make the algorithm slower. This effect can be caused by:

algorithmic penalty - due to the nature of the algorithm. The more inherently sequential the algorithm, the less capable it will be as a parallel algorithm,

implementation penalty - the time a processor spends doing non-useful computation. Two processors solving a problem twice as fast involves them spending all of their time working on the problem, which we know is virtually impossible to achieve.

4.5.1 Performance Metrics

The solution time of a parallel implementation is a simple way of evaluating it. If more detailed analysis of a parallel implementation is needed, a range of metrics must be used to compare and contrast different approaches.

Speed-up

The ratio of time taken on a uniprocessor system to the time taken on the multiprocessor system can be expressed as

$$\text{speed-up} = \frac{\text{elapsed time on uniprocessor system}}{\text{elapsed time on multiprocessor system}} \tag{4.1}$$

The term *linear-speedup* is used when the solution time on a n-processor system is n times faster than the solution on the uniprocessor. We say a parallel implementation has suffered *speed-down* if it is slower than the uniprocessor implementation. If the multiprocessor version is faster than linear speed-up it is said to have *super-linear speed-up*. This would seem an impossible scenario but it can be caused by the removal of paging due to the large memory needed by a problem, combined with the fact that more than one processor is working on the problem. Another case where super-linear speed-up might occur is due to the better use of cache, caused by the improved coherency of a parallel implementation. There are two ways to obtain the time taken by a uniprocessor, by timing:

1. an optimised sequential algorithm on a single processor or,

2. a parallel implementation on one processor.

The first method is applied in most cases where comparison is used, as normally the parallel algorithm is not as efficient as the sequential version. When determining the scalability, using the second timing would give a more meaningful result.

Efficiency

The efficiency measures the average performance of just one of the processors in a parallel computation. This then enables us to determine the overheads caused by the parallel algorithm. It is obtained by

$$\text{efficiency} = \frac{\text{speed-up} \times 100}{\text{number of processors}} \tag{4.2}$$

From this we can gauge what percentage of the total time each processor spends on the actual computation of the problem, and therefore how scalable it might be.

Using all these performance metrics, one should be able to predict the ideal number of processors that should be used with any parallel algorithm. This optimum is not solely dependent on the algorithm used, but also on the problem one is trying to solve. However, by examining graphs of the speed-up for increasing numbers of processors, it is possible to see when adding more processors would definitely be detrimental.

4.5.2 Models for Parallel Algorithms

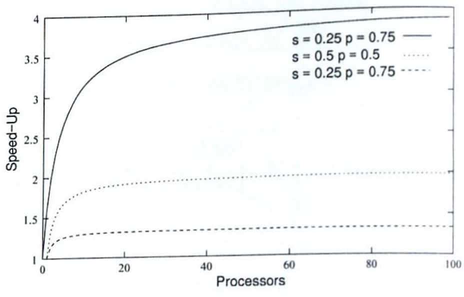
Over the years, various models have been proposed for the estimated performance of parallel algorithms. In 1967 Amdahl developed what is now called "Amdahl's law", which attempts to give a upper bound on the speed-up of an algorithm due to its properties. His model is as follows:

$$\text{maximum speed-up} = \frac{(s+p)}{s+\frac{p}{n}}, \tag{4.3}$$

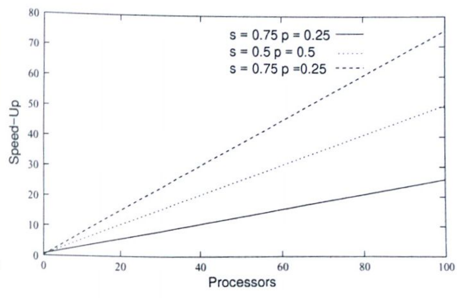
where s is the time spent on sequential parts, p the time spent on parallel parts and n the number of processing elements. The total time taken by an algorithm on a single processor is $s + p = 1$. Amdahl's law gives some rather negative forecasts for large numbers of processors. Gustafson [49] proposed that the size of a problem is hardly ever independent of the number of processors as assumed by Amdahl's law. His model is represented by the following

$$\text{maximum speed-up} = \frac{(s+pn)}{s+p}. \tag{4.4}$$

Gustafson's model predicts a nearly linear speed-up when the problem size is increased by adding more processing elements. However, it does not apply in all cases, since usually the problem size is predetermined, and just the processing time needs to be reduced. This situation is typical of global illumination problems. Thus for fixed size problems, Amdahl's law is probably more applicable.



(a) Amdahl



(b) Gustafson

Figure 4.4: Maxmum speed-up using Amdahl and Gustafson's laws

4.6 Problem Decomposition

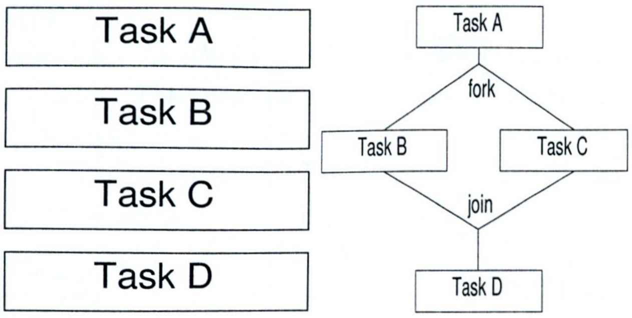
There are two principal ways a parallel problem can be broken up, either by exploiting the inherent parallelism in the algorithm itself, called *algorithmic decomposition*, or by using the fact that the problem domain can be split up into many parts to which the algorithm can separately be applied. This is called *domain decomposition*.

4.6.1 Algorithmic Decomposition

The algorithmic decomposition of a program involves rewriting the original algorithm or, better still, creating a completely new algorithm. This can be quite an involved process and requires explicit knowledge of the parallel features of the system. The decomposition can be quite low level, for instance using MMX or AltiVec instructions to exploit instruction level parallelism, known as *dataflow*. A *fork and join* method is feasible if you have something like SMP (see Figure 4.5). Despite the difficulties involved, significant speed-ups can be achieved with these types of decompositions. Unfortunately, not all problems can be solved efficiently by such parallel algorithms.

4.6.2 Domain Decomposition

Domain decomposition is far easier to apply than algorithmic decomposition as it does not involve changing the algorithm in any way. Instead it subdivides the domain of the problem into small segments which can be solved independently. A simple example of this is the problem of integrating a function. Note that



(a) Serial (b) Fork and Join

Figure 4.5: Algorithmic Decomposition

integration in two dimensions determines the area under a curve. It is therefore obvious that breaking this large curve up into n smaller ones and determining the area under each of these curves, then finally summing these areas, determines the area under the original curve. The domain decomposition method can be broken into two main methods from which hybrids [103, 107, 105] can be formed. These methods are:

Data Driven - This method involves breaking up the data between the PE's involved such that each one gets a portion of the total data. This method unfortunately requires either prior knowledge of the data or a preprocessing pass to gain such knowledge. Since the data is divided among the processors, tasks must now be migrated to the PE that has the data required for the task.

Demand Driven - The demand driven method divides the work and not the data between the PEs. Thus each one is no longer stuck with a given portion of the data. This has the effect of requiring that the data be moved around if it does not fit entirely on a single PE.

4.7 Scheduling Issues

The key to the efficient use of these methods is in scheduling, so that the communication overheads are minimised in such a way that no PE is left idle. This is often easier said than done. There are two categories of scheduling algorithms:

static - The assignment of tasks or data is done before the program begins at compile time. This method assumes that knowledge of the complexity of each task is known at compile time.

dynamic - The data or tasks are redistributed during the execution of the program. This is achieved by a method called *load balancing*, whereby tasks are transfered from heavily loaded PEs to lightly loaded ones.

4.8 Parallel Methods and Global Illumination

Ray-based global illumination methods such as path tracing appear to be embarrisingly parallel, making them ideal candidates for use with parallel computation. The obvious strategy of dividing the image space into a number of segments and assigning these to each PE works very well. This can be done using static or dynamic [55] load balancing schemes. Sometimes the memory needed to store a scene and its intermediate data is far too large. In order to solve this problem, the scene can be divided among a number of PEs. This creates a data driven rendering scheme [48, 108, 104, 9]. The main problem associated with data driven schemes is that PEs can remain idle for long periods of time, because the region of the scene they hold is only accessed infrequently, such as areas which are in shadow. Areas which are visible or contain lights are accessed much more frequently, making it hard to balance the workload. Thus there is a trade-off between having equal amounts of scene and having equal processing loads. Cost prediction schemes have been devised [109] so that scenes can be divided based on their workload. To remedy this situation, hybrid schemes have also evolved which contain both data and demand driven parallel schemes [106]. To minimise the communication overheads, caches [104, 58], level of detail and coherence [126] have all been used. These either eliminate the communication or reduce the amount of data that must be sent.

For FEM based global illumination methods, data driven schemes are most widely used such as virtual walls [4] and visibility masks [4]. However, solutions which use parallel versions of the matrix methods such as Jacobi iteration, Southwell relaxation [47] or group itterative methods [111] provide an alternative means of acceleration. There are also parallel methods based on hierarchial radiosity [120].

Chapter 5

Algorithms

5.1 Introduction

This chapter presents a series of tools and adaptive schemes which are used to solve global illumination problems. The initial section presents a rendering framework, which allows various algorithms to be easily implemented and used to help solve the many complex problems involved. This is then followed by a description of some "intelligent" sampling schemes, which sample the parameter domain of the function they are used with. These create better sample sets for use with integration and density estimation problems. The final section presents some density estimation methods, which are used to reconstruct illumination information that has been stored as a series of particles or samples distributed, using MCMC, throughout the environment.

5.2 Framework

Very often, researchers are only interested in a small area within graphics, such as surface modelling or scene query acceleration. However, in order to start their research, a substantial effort has to be put into creating a rendering system capable of supporting their needs. They can either use an existing system or create a new one. Both of these approaches seem to require significant time, in either constructing the system or learning how to use and change an existing one. Another problem with creating a rendering component is caused by the use of statistics. This makes it hard to determine if the algorithm is working correctly since the results are stochastic. One approach to checking this is to import test scenes from other rendering systems and compare the results. Visual comparison is not very accurate and often the effects caused by a bug don't show up until much later on even after the algorithm was thought to be working. Comparing results with others is often very difficult and generally requires actually recoding other algorithms so that they fit in with your approach. This introduces bugs and so could end up being different from that of the original implementation. Since these are not the focus of the creator's research, very little effort is extended towards these algorithms. What's needed is a framework for constructing a rendering system that separates all these components so that they can be swapped in and out of the system without the framework having to be aware of this. This way algorithms can easily be compared and tested. Also an extensive set of tools are needed for visualising various component types, such as scene acceleration structures and ray directions. This way the user can easily see what is happening as the program is running, without having to use a debugger to view the data structures and interpret the data themselves. This framework must not restrict or force the user to think in a certain way or do significant extra work just to make their algorithm fit into the framework. They must be able to code in the same manner as before. Thus the user should only have to understand the part of the framework concerned with their area, rather than have to know everything about the entire framework. The code re-use made possible by the object oriented and modular techniques, are a very important part of the framework and help with this task.

On another note, despite all efforts to find models and techniques which are suitable for all scenarios, these have not been developed as yet. Therefore a framework should provide a means of allowing all methods and models to coexist simultaneously. This is so that a system or user can choose the appropriate method or model to suit a specific scenario. Many frameworks don't split methods of solution from the problems they are trying to solve. A classic example of this is the use of integration techniques in radiosity or Monte Carlo path tracing programs where the function to be integrated is embedded in the integrator. Thus each time you create a new method you must also recreate the necessary parts of the function. This limits the areas where you can use the method and it makes testing far harder. However, for the following reasons it is sometimes desirable to separate these. Firstly the increased visibility of mathematical components make it easier to solve specific problems. Secondly, separating the description from the solution contributes to smaller, reusable components, which can be combined and tested with greater ease.

What follows describes a rendering framework called EFFIGI (Efficient Framework For Implementing Global Illumination) [77], that uses interfaces which express both geometric concepts alongside mathematical ones, using object-oriented and component object methods. It also handles other concerns that potential users might have, such as object description and scene acceleration. The framework created differs from most other frameworks by mapping many global illumination concepts to mathematical ones rather than the reverse, making it far easier to implement mathematical techniques. This is because the physics have already be translated into mathematical models so they can be simulated on a computer. However, translating mathematical techniques into their physical counter parts is far harder and also very specific to the physical scenario thus reducing their applicability.

5.2.1 Previous Work

Many mathematical models are used in image synthesis, the rendering equation [63] being one example. These mathematical models are solved by a variety of techniques. By simplifying these, easier and faster methods of solution may be found. For instance the radiosity equation [18] is a simplified form of the rendering equation that can be solved using finite element methods such as Galerkin [137] or point collocation [19] methods. This simplification limits the types of environment that can be described. More general descriptions like the rendering equation (see Figure 5.1) need more general methods of solution such as Monte Carlo integration. Some of the methods come directly from numerical techniques such as Metropolis [130] or Galerkin methods, others such as bi-directional path tracing [129, 69] are derived from the nature of ray paths. Most methods have the common property that they are described using mathematics.

Various architectures have been devised to handle this situation. Some have been tied to a specific technique [23, 67, 117, 134, 131], others have been general enough to allow a variety of rendering techniques. The Cornell "testbed for image synthesis" [124] is a toolbox that can be used to construct a rendering package but is not object-oriented. Glassners "Spectrum" architecture [43] is an object-oriented framework based on signal processing. The Vision system [122] is capable of most rendering methods as is the RenderPark system [11].

5.2.2 Design

The basic design of the frame work is to model the mathematical concepts used to solve global illumination problems. It is done this way because it is easier to map global illumination descriptions into mathematical ideas, than to map mathematical methods into physical concepts. The reason being that mathematical abstractions, such as adaptive integration methods, were derived using the abstract mathematical descriptions and hence do not appear to have intuitive physical interpretations. This also improves the generality and promotes reuse of methods as they are no longer designed to be used within one area, because they retain their original mathematical generality. The use of mathematical ideas also provides far greater flexibility and brings intuitiveness to the framework since everybody understands mathematics to some degree. For instance one could combine functions represented in the framework by addition, subtraction, multiplication etc. Then this combination could be integrated. In fact, this is how many complex functions such as BSDF's are built within the framework.

The design of the EFFIGI framework is motivated by several key factors. It is fully object-oriented, thus enabling increased flexibility and code re-use. The abstractions are based not on physical concepts such as radiance and radiosity but instead on their mathematical foundations. In order to make development easier there is a means of testing components. The time taken to generate a picture is very important, so speed has been taken into consideration throughout all phases of the design process.

Object Oriented Design

Object oriented techniques [26] are used to decompose the rendering process into simple manageable parts. The interfaces that arise from this fit into four main groups:

- *mathematical* for sample generation, function evaluation and integration.
- *scene and object management* for ray queries, meshing and path generation.
- *data storage* to support data structures such as linked lists, k-d trees [12] and vectors.
- *setup* for object initialisation and setup.

Within each of these groups, various sets of interfaces are provided. For a full list of interfaces see Table 5.1.

Math	Scene	Setup	Data
`ISampler`	`IIntersect`	`IRenderer`	`IContainer`
`IFunction`	`IShape`	`IInterface`	`IVector`
`IIntegrator`	`IMesh`	`IInitialise`	`ITree`
`IGenerator`	`IRay`		`IList`
`IRootFinder`	`IIlluminationMap`		
`IWarp`	`IPathGenerator`		
	`IEvent`		
	`IEventStore`		

Table 5.1: Interfaces in Framework

$$
\begin{aligned}
L(\vec{\omega_i}, y) \;=\;& \underbrace{L_\epsilon(\vec{\omega_{i_x}}, x)}_{\textit{IFunction}} + \\
& \underbrace{\int_\Omega \overbrace{\overbrace{f(\vec{\omega_{i_x}} \to \vec{\omega_{o_x}})}^{\textit{IFunction}}\, L(\vec{\omega_{i_x}}, x)}^{\textit{IFunction}} \cos\theta_{i_x}\, d\vec{\omega_{i_x}}}_{\textit{IIntegrator} \text{ and } \textit{IFunction}}
\end{aligned}
$$

Figure 5.1: The Rendering Equation

Mathematical Foundation

The abstraction of mathematical concepts provides generic access to various algorithms, allowing a "pick-and-mix" approach to choosing a suitable technique. This abstraction is important in image synthesis, because many models use mathematically-based methods. For example, the light propagation model used in rendering utilises sampling and integration. EFFIGI provides explicit support for these ideas, by translating concepts such as functions, integration and sampling into interfaces. The learning curve for the users of such a framework will not be steep, since users are typically well-aware of the mathematical ideas involved.

The most common interfaces used are the *IIntegrator* and *IFunction* interfaces. These are usually used to represent the various parts of our image synthesis model. The *IFunction* interface can be used to represent surface reflection models or to evaluate ray paths. When used by an *IIntegrator* the total light reaching a point can be estimated. Figure 5.1 shows how this can be encapsulated within another function interface to represent the rendering equation.

The parameters of the inner most *function* represent the direction (θ, ϕ) that the ray goes when it strikes a surface as in the case shown. They are equally likely to represent a surface and coordinates on it (s, u, v), if a surface form of the rendering equation is used. This configuration represents a distributed ray tracer since the function is recursive. A more efficient implementation would expand the inner function and use it to represent n bounces, where n is the first parameter, thus creating a path tracer. To allow this the function interface provides methods which disclose this information. The *IFunction* interface is given in Figure 5.2. This interface enables functions to be evaluated, and provides information describing the limits of the domain of the function and the range of the result. The *IIntegrator* interface is shown in Figure 5.3. It provides a means

```
struct IFunction : IUnknown
{
 virtual integer GetDimension(void) const = 0;
 virtual real Evaluate(const real *p_parameters) = 0;
 virtual void GetDomain(real *p_min,real *p_max) const = 0;
 virtual void GetRange(real *p_min,real *p_max) const = 0;
};
```

Figure 5.2: IFunction interface

```
struct IIntegrator : IUnknown
{
 virtual void SetSamples(integer samples) = 0;
 virtual real Integrate(const real *p_min,
                        const real *p_max,IFunction *p_func) = 0;
};
```

Figure 5.3: IIntegrator interface

of evaluating an integral over a region in the functions domain, and setting the number of samples needed.

Testing Facilities

An important feature of the framework is its facility to test components for compliance with a given specification. For instance, a Bi-directional Scattering Distribution Function (BSDF) should be physically plausible; a scene acceleration scheme should support all the required features, and provide the same intersection information as another scheme given the same data; an interface should produce valid responses to a specified set of inputs. Previously, it was necessary to run components with a complete setup, and then attempt to identify artefacts in the output. Now users can run the testing components with default setups, allowing rapid identification and location of errors. This speeds up the development process significantly. It is also possible for users to quickly ensure that the interfaces provided can support any new components that they may wish to add to the framework. There are also visualisation tools for viewing the rendering process. For instance Figure 5.4 shows a component that allows you to see the rays as they are fired either from lights or the view plane.

Scripting Language

In order to create an easy, high level way to configure the system a very basic scripting language was created. This language basically enables the creation, destruction, configuration and linking of the various components used in the

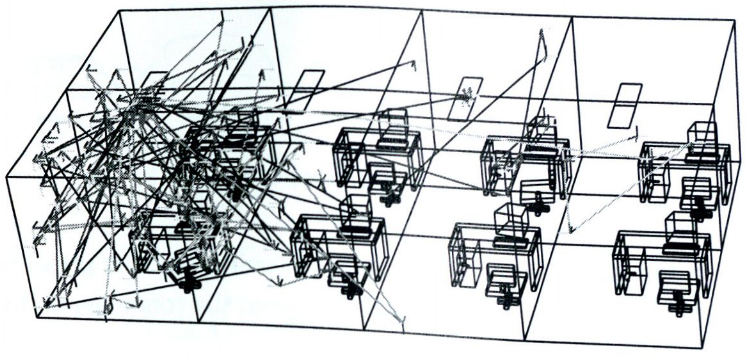


Figure 5.4: Ray Viewer

system. The language bears some resemblance to C++. It can be used to modify any component, enabling scenes to be setup or scene files from other rendering packages to be loaded using a component that provides this. In this way users can try various setups very quickly and easily with out having to compile and build a setup using lower level constructs. This makes setting up diagnostic and analysis setups very easy. Since speed is not of the essence during setup and configuration the language can be very high level and interpreted which reduces the number of errors and bugs that occur. The scripting language means the user no longer has to compile their components with the entire framework, but can instead create it separately as a shared library. Thus relinquishing the need to try and fit their code into the build hierarchy of the code base. This simplifies the integration of components and speeds up the compilation time.

5.2.3 Implementation

The *Component Object Model* [68, 110] was used in EFFIGI to provide a consistent interface for creation and destruction of components, and to provide Run Time Type Information (RTTI). For an in depth discussion of COM see [68] and [110]. It allows the use of shared libraries or DLLs[1] which gives the framework the ability to have a "plug-in" style component architecture without any extra work by the programmer.

To date, EFFIGI has been used to implement many of the major rendering methods in use today. It can be configured to use a multitude of different techniques with ease. The framework can be used in two ways, via a simple scripting language or by using a programming language that supports COM such as C++. What follows are some example setups using the scripting language that illus-

[1]In UNIX they are called shared libraries in Windows dynamic link libraries or DLL's. These enable the sharing of compiled code by allowing libraries to be dynamically loaded at runtime.

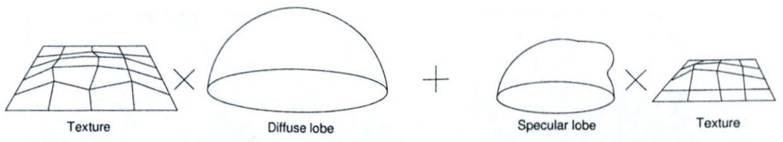


Figure 5.5: BRDF composition

trate some typical uses of the framework. Each configuration is presented with a diagram showing the communication links between the components that make up the method.

Implementation of a BRDF

The implementation of a BRDF using the framework is very different from many of the other frameworks available. Because the framework is expressed using functions, to create a BRDF one needs to combine a series of functions together. For instance, to create a Phong BRDF [71] a diffuse cosine lobe must be combined with a specular one; To add texture to this, the diffuse cosine lobe is multiplied by a texture function, e.g. a linearly interpolated function taking the surface coordinates (u, v) as parameters. Then another function may be used to perturb the specular power according to the surface position (see Figure 5.5). This provides a very expressive way of creating BRDFs, allowing not only the developer to create many custom BRDF's, but also the user. All this functionality is provided by a series of components which support the *IFunction* interface.

Path Tracer

The path tracer [63] is constructed from a few basic components as shown in Figure 5.6. Direct Lighting is achieved using an *IFunction* interface which evaluates rays from the light to the point in question. This is then called from inside the path evaluator. Each box represents a component which can be substituted at run time, and thus a variety of configurations are possible. The picture shown in Figure 5.7 was rendered using a VEGAS [80] style integrator with a directional path generator and a surface to surface direct lighting function.

Photon Map

A Photon Map [61] is a preprocess which is used to create illumination maps (see Figure 5.8). These can be used with the path tracer configuration just described by using a modified path evaluator. These path evaluators query the illumination maps at a given point, obtaining an estimate of the incoming radiance at that point. The photon generator can use all the samplers and warping

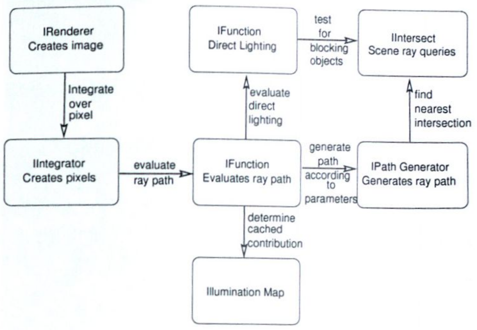


Figure 5.6: Path Tracer Configuration

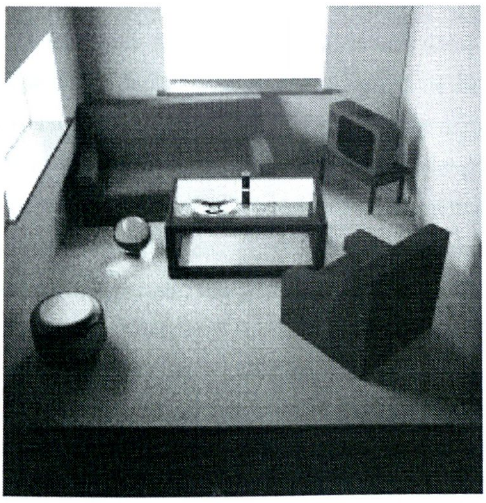


Figure 5.7: Path Tracer Example Picture

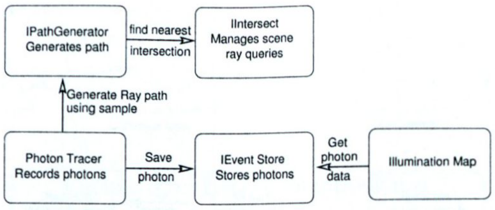


Figure 5.8: Photon Tracer Configuration

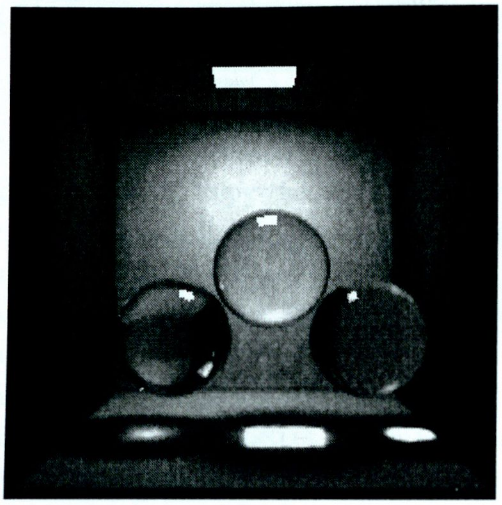


Figure 5.9: Photon Tracer Example Picture

schemes just like a path tracer, storing photons using an *IEventStore* interface. The component used to store the photons can use a k-d tree, quad tree or any other suitable structure, and might also directly support an *IIlluminationMap* interface. Alternatively, it can provide some facility to convert it to a suitable structure for some other component which would provide the illumination map during the rendering phase. Figure 5.9 shows a Photon Map used in conjunction with a path tracer, which in turn uses a Mean Sample Monte Carlo integrator [64] with a random sampler. The photons were generated from the light source using a Metropolis sampler which sampled the emission function $L_e(u, v, \lambda, \vec{\omega})$ of the light source to generate a sample set. A Metropolis sampler is a component based on the *ISampler* interface. It takes an *IFunction* interface and generates a set of samples, using the function as a basis for the PDF, this is done using the Metropolis [64, 130] method.

Irradiance Map

An Irradiance Map [134] is a modified path evaluator, which caches radiance information in an illumination map, also using an *IEventStore* interface. It is

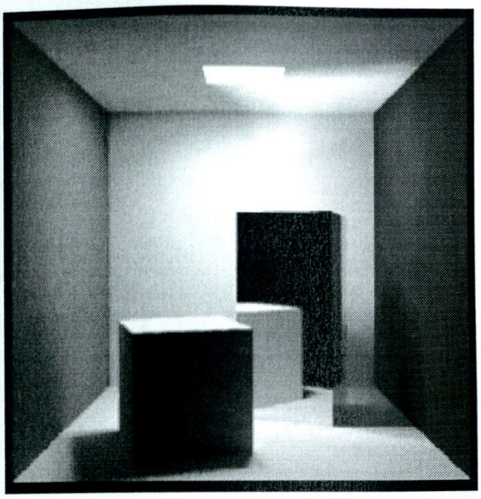


Figure 5.10: Irradiance Map Example Picture

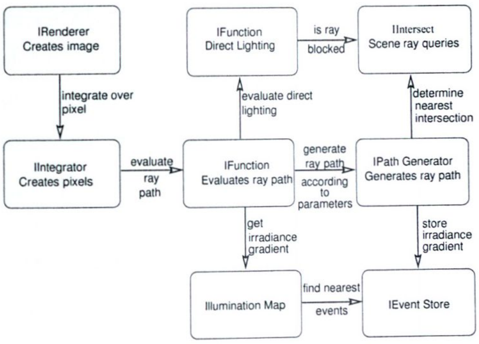


Figure 5.11: Irradiance Map Configuration

possible to re-use this interface, because the *IEvent* structure used with the event store dictates what is stored and what the information relates to. The *IEvent* interface provides a way to store information. By changing these events, different types of information can be stored, such as irradiance gradients or photons. Unlike producing a photon map, the irradiance map is not created during a preprocess but by a modified path evaluator (see Figure 5.11). Shown in Figure 5.10 is a picture generated using an Irradiance Map style path evaluator and a Mean Sample Monte Carlo integrator using a Hammersly [100] quasi-random sample set.

Radiosity

Radiosity [19] is implemented as a preprocess, as in the Photon Map method. Similarly radiosity is stored in a component, this time using the *IMesh* interface. This interface supports the ideas of patches from which form factors can be cal-

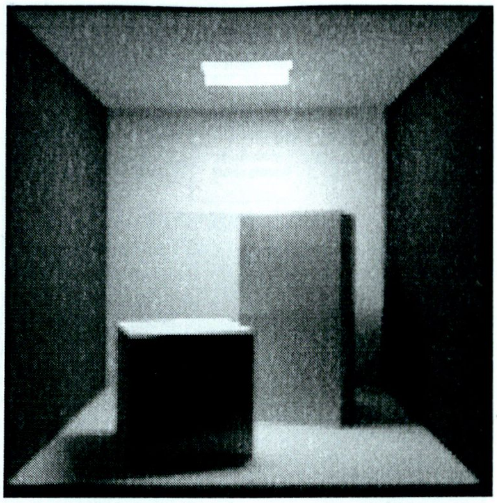


Figure 5.12: Radiosity Example Picture

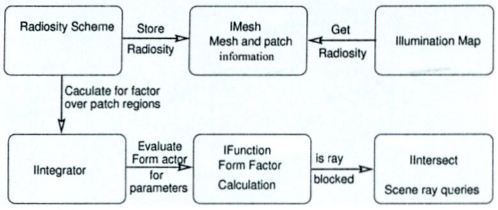


Figure 5.13: Radiosity Configuration

culated and radiosity stored. The component may support an *IlluminationMap* interface, or it may provide the ability to output a texture, or something that can be used by an illumination map component. The form factor is calculated using a component that supports an *IFunction* interface that takes two surfaces as properties, the parameters of which are the (u, v) surface coordinates on each surface. This enables the use of various integration schemes (see Figure 5.13 for the configuration used). Figure 5.12 is a radiosity illumination map rendered using a Mean Sample integrator with random sampling. The radiosity structure used a Metropolis sampler to integrate the form factor function using the visibility function as a basis for the PDF.

5.2.4 Scene management

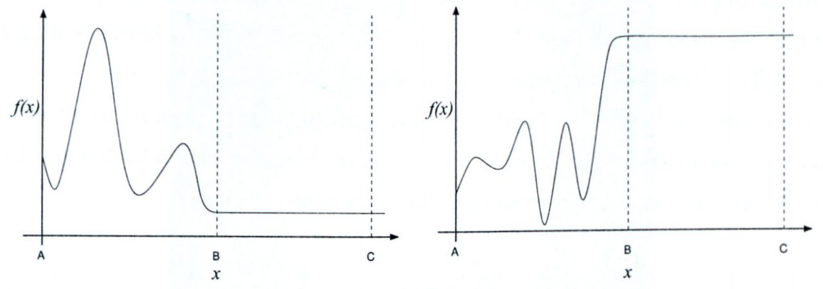
The scene and object management interfaces allow a lot of modularity and the construction of objects is quite similar to that of the ray tracing framework by Arvo and Kirk [67]. Many different scene acceleration methods can be combined in a variety of ways, so as to construct a more optimal setup. For instance, SEADS grids [38] and octree [42] schemes can be used together or just over individual objects or areas where needed. The visualisation facilities are very beneficial for examining exactly what a scene acceleration scheme is doing (see Figure 5.4). Testing for faults is facilitated by comparing results from two schemes. Objects can be constructed by grouping sub-objects together using the scene acceleration structures or via other components, which can then be assigned various properties and sampling schemes.

5.2.5 Use of Framework

All the methods that follow were developed using the EFFIGI framework. This allowed comparisons with other methods and made the development process quicker, since only the new modules needed to be implemented, thus reducing the amount of coding and debugging necessary. It also facilitated the use of new techniques in ways which would never have been possible without the framework, for example Metropolis particle tracing, the VEGAS and stratified integration techniques as well as the method used for describing BRDFs, which are presented in this document. This was facilitated by the generic interfaces provided. The development process was simplified by the ability to test the new techniques on simple functions, not necessarily related to global illumination for which the results were known. This means that any new technique created can be tested on simple functions, thus permitting greater confidence in the results that it produces, which is impossible with any other framework.

5.3 Adaptive Integration Schemes

This section presents two different adaptive integration schemes, based on the stratified and importance variance reduction schemes presented earlier. The stratified schemes would appear to be the best, since they correctly handle areas of high variance (see Figure 5.14), because areas of high value are not necessarily the most difficult regions to integrate. However, to date their use in computer graphics has been limited, and importance sampling is more commonly used. This is probably because importance sampling, which has been very successfully used for sampling BRDFs, is far easier to implement. Stratified sampling has been used to split ray paths and break BRDFs into a diffuse and specular part for use with illumination [6, 20, 102], irradiance [134] and photon maps [60]. Adaptive schemes seem to be a good idea for global illumination problems because of the uneven distribution of work and variance. These schemes, if set up properly can hopefully detect and cater for this unevenness, thus producing a better overall result.



(a) Effectively Sampled (b) Ineffectively Sampled

Figure 5.14: Two cases which highlight the problem with importance sampling

5.4 Optimisations for Adaptive Schemes

Unfortunately, the high dimensions experienced in global illumination makes the detection of regions of high variance difficult. In order to address this, the standard algorithms must get assistance in the form of prior information about the problem domain. Such information includes importance sampling schemes for BRDFs, the relative importance of each parameter and so on.

5.4.1 Simplification

The biggest problem with using adaptive sampling schemes is that they can incorrectly predict areas of importance. This can occur because they are only estimating areas of relevance using the limited set of information obtained during previous computations. In order to optimise such schemes, it has to be made easier for the adaptive schemes to single out regions of the parameter domain which are important, or need more work. This is done by re-parameterising our lighting function from the complex forms commonly used in many of today's global illumination schemes to a simpler form. These complex algorithms tend to gather as much information as possible from each parameterisation, by trying to evaluate many paths in one query. This confuses adaptive algorithms, but can be solved by using functions that evaluate only one path at a time, instead of combining a series of paths. Techniques such as direct lighting on each ray hit can be used. Now rather than multiple effects appearing in one evaluation, only one possible effect can appear, leading to clear differences between areas which contribute and those that don't (see Figure 5.15). This does not make the algorithm less efficient, as now fewer rays are shot and less rays will be used on paths with many bounces, whereas in normal path tracing equal effort is put into sampling secondary and subsequent bounces, even though they do

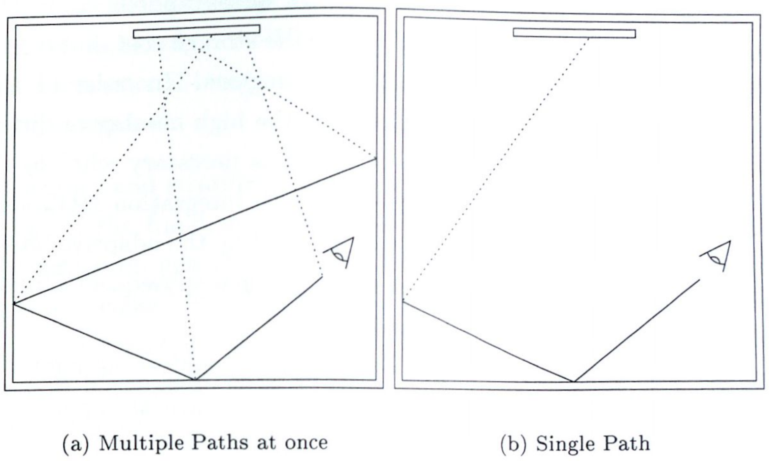

(a) Multiple Paths at once (b) Single Path

Figure 5.15: Reducing the number of paths evaluated

not contribute as much to the overall solution. The simpler formulation makes it easier to perform importance sampling, since each query evaluates only one path and therefore does a similar amount of work. It might seem that these ray parameterisations are reverting back to less efficient forms, but this is not necessarily true if the time to evaluate the function is factored into the sampling strategy. The situation is further improved now that evaluations are cheaper, since more evaluations can be made in less time from which more information about the parameter space can be derived. This in turn produces a better PDF estimate. Adaptive schemes suffer from the need to store potentially large volumes of data, which is exaggerated by global illumination problems due to the number of dimensions required to parameterise ray paths. The schemes presented below solve this problem by only storing information for limited sets of dimensions. An alternative way of reducing the storage cost is to reduce the dimensionality of the parameter space by re-mapping parameters. This unfortunately can only be done to a limited extent, although the reduction in storage costs are usually very good. A side effect of doing this is that now information about the problem domain has been merged, precisely the situation we wish to avoid. An example of this approach that can be used for surface to surface path tracing, is to merge the surface selection parameter s with either the u or v parameter used to determine a position on the surface giving:

$$\begin{aligned} s &= \text{int}\,(us) \\ u &= us - s, \end{aligned}$$

which is a very good optimisation for use with quasi-random methods. The problem of dimensionality does not only effect the storage cost, but also makes it harder for the algorithm to find important regions. In order to alleviate the constraints on adaptive schemes caused by the high number of dimensions associated with our problem, a mixed approach is necessary whereby density estimation methods are used in conjunction with integration methods. The first two or three ray bounces are therefore handled by the adaptive integration schemes, which need a large amount of accuracy, then subsequent illumination is handled by density estimation methods.

As a last resort, but perhaps the most effective solution, user intervention may be used to highlight parameters and regions of potential importance.

5.4.2 Previous work

Keller demonstrated the use of quasi-random sampling [66, 53, 54, 65] with particle (or potential) based methods and with radiosity. Kajiya introduced two dimensional stratification schemes in his landmark paper about path tracing [63]. The super sampling [82] strategies discussed by Mitchell show the advantages of stratified sampling schemes. The VEGAS [79, 80] method used in part by Dutré [34] applies some stratification techniques. Shirley's work on Direct Lighting [114, 119, 118] demonstrates suitable sampling schemes and divides the scene into two parts namely the light sources and all other objects. These techniques are either limited in the dimensions that they can stratify or have non-adaptive stratification schemes. The Miser [99, 100] algorithm remediates these limitations but has never been applied to the problem of global illumination. A number of methods to do this have been devised such as a 5D Tree [72], adaptive probability functions [34] and importance driven path tracing [59]. The use of these techniques has been restricted to estimating the local contribution of a light source, or constructing an adaptive PDF for the BSDF of surfaces in the scene.

5.5 VEGAS Adaptive Integration Algorithm

This method uses an adaptive PDF to choose the most important ray paths coming from the eye. The algorithm is a probabilistic "gathering" algorithm based on path tracing. Two sampling strategies are used: The first uses the BRDF of the surfaces to determine an importance weighting for the outgoing ray directions. The second uses an adaptive PDF to weight segments of the paths. Thus there is both a local importance sampling scheme and a global one. The PDF for the global scheme is initialised as a uniform sampling function but is refined during the execution of the algorithm. The construction of the refined PDFs is based on the VEGAS algorithm [80, 79], which allows for adaptive multidimensional integration. In this manner the algorithm progressively "learns to integrate".

Importance and stratified sampling methods unfortunately require detailed knowledge of the function's behaviour to be of use. However, it is possible to use information generated in a previous integration to reduce the variance in subsequent cases. The VEGAS algorithm [80, 79] is a multi-pass adaptive sampling algorithm that also provides support for both stratified and importance sampling. It is normally used in particle physics simulations but the following will demonstrate its usefulness in image synthesis. The algorithm adaptively constructs a multidimensional weight function w that is separable:

$$p \propto w(x, y, z, \ldots) = w_x(x)w_y(y)w_z(z)\ldots, \tag{5.1}$$

where p is the PDF and $w_x, w_y, w_z \ldots$ are the individual weight functions for each dimension. It is a multi-pass method because, in order to construct the weight function, coarse initial estimates of the separate functions need to be found. Further passes are made to refine these weight functions, thus improving the overall estimate. The optimal separable weight function w_x can be shown to be

$$w_x(x) \propto \left[\int dy \int dz \ldots \frac{f^2(x, y, z, \ldots)}{w_y(y)w_z(z)\ldots}\right]^{\frac{1}{2}}. \tag{5.2}$$

Equation 5.2 presents a means for creating each separable weight function using Monte Carlo estimation methods. Each dimension of the domain of integration is initially divided into K equal sized partitions, giving a total of Kd bins, this minimises the storage costs. As samples are taken from the function the corresponding value is deposited into one of the K bins in each dimension. After each iteration the bins are reassigned according to either the importance or variance of each partition, enabling partitions with a higher contribution or variance to get more samples. Importance samples are generated by choosing a bin at random in which a sample is then generated. Thus, initially each sample is chosen uniformly with respect to the integration domain, but as the bins are altered on each iteration they are gradually drawn from the function. On the other hand, in stratified sampling an equal number of samples are drawn from each bin, so K_i^n samples are needed where K_i is the number of partitions in each dimension and n is the dimension of the function. The PDF is a stepwise function which is initially uniform over the whole domain. It is then updated after each iteration by adjusting the size of the bins or step functions and, since each interval has a equal chance of being selected ($\frac{1}{N}$), more samples are

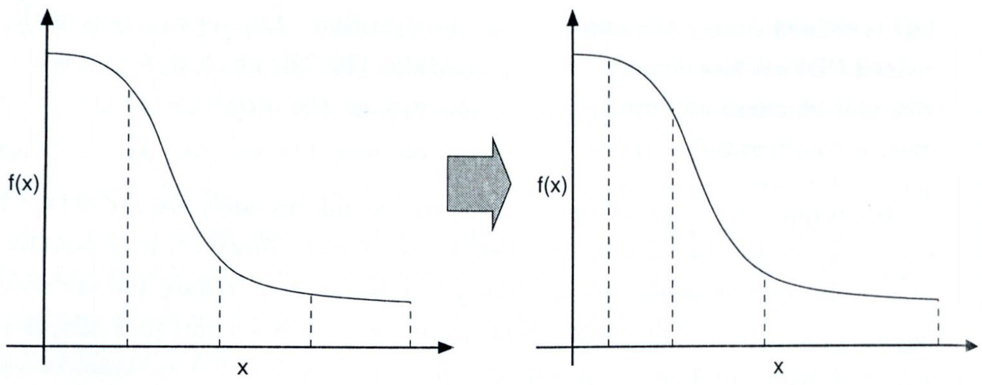


Figure 5.16: Initial PDF (left) and PDF after a number of iterations (right)

concentrated in the important areas (see Figure 5.16). The probability is given by

$$p(x) = \frac{1}{K\Delta x_i}, \tag{5.3}$$

where K is the number of divisions and Δx_i is the size of the i^{th} division (size being the length in 2D, area in 3D etc.). This type of control over the PDF allows it to adapt more closely to the function it is sampling since the bins can move as well as change size to adapt to the function. This adaptive scheme is hard to implement in more than one dimension because adjusting the positions and sizes of the bins becomes increasingly difficult, due to the effect on neighbouring bins. A kd-tree [12] was used by Dutre [34] to create a similar structure for 2D sampling. However only one division can be altered on the first iteration. When a good position is found this division is fixed, and then child divisions are created and modified.

Another way to generate an adaptive PDF is to use a unit hypercube. Here, every sample taken is recorded and later used to construct a cumulative density function which is then used to generate samples [59]. However, the extent to which this can adapt is determined by the number of grid increments N along each axis. N in turn is limited by the total number of integrand evaluations M allowed per iteration:

$$M = 2N^n, \tag{5.4}$$

where n is the dimensionality. This is a serious limitation with the higher dimensions required in global illumination problems.

5.5.1 Algorithm

The problem in global illumination is given in the rendering equation. To solve this equation a number of discrete paths are chosen at random. This is in most

```
WHILE $i \leq N$
        IF $i$ % $update = 0$
            FOR EACH weight function
                IF1D OR 2D
                    setup stepwise using current estimates
                ELSE IF 3D or above
                    setup hypercube using current estimates
                END IF
            END FOR
        END IF
        Generate sample according to global weight function
        FOR EACH weight function $i$
                store result in appropriate bin
        END FOR
END WHILE
```

Figure 5.17: Adaptive sampling algorithm

situations very inefficient, as it is better to sample paths which contribute more, or to sample in areas which decrease the variance. For this reason, most implementations of path tracing and similar Monte Carlo algorithms use importance sampling, exploiting knowledge of BRDF to sample ray directions. This algorithm helps path tracing methods to sample paths more effectively, by taking into account the paths where the incoming flux is greater.

The algorithm uses a combination of the methods described previously. A separable weight function is constructed as in the VEGAS algorithm. However, rather than using a one dimensional weight function for each dimension, a series of multidimensional functions are created. These are used to construct the overall multidimensional weight function, which is then used to sample the paths. Since the most prominent errors in path tracing occur on the first and second bounce, these are the areas for which we generate our adaptive PDF. The user can configure the dimensionality of the sub-functions used to construct the multidimensional weight function and limit its total dimension. The following is a typical grouping of variables for which sub-functions may be defined:

$$\{ \overbrace{\lambda}^{\text{wavelength}}, \underbrace{(u,v)}_{\text{film}}, \overbrace{(u,v)}^{\text{lens}}, \underbrace{(u,v)}_{\text{lightsource}}, \overbrace{(\theta,\phi)}^{1^{st}\text{bounce}} \}$$

Thus, we have a series of one or multi-dimensional PDFs, which the algorithm generates and uses to sample segments of the path. These PDFs are created using a hypercube for dimensions greater than two, or the stepwise function for lower dimensions. The algorithm is shown in Figure 5.17.

As mentioned the algorithm differs from the original VEGAS algorithm in that it is not constrained to use only one dimensional sub-PDFs but a series of PDFs of differing dimensions. The equations for this new form are a simple extension to the original proofs for the VEGAS algorithm. The optimal densities for a separable PDF are given by

$$p(\omega_a^1, \ldots, \omega_a^n, \omega_b^1, \ldots, \omega_b^n) = p(\omega_a^1, \ldots, \omega_a^n) p(\omega_b^1, \ldots, \omega_b^n), \tag{5.5}$$

where ω_a^i is the i^{th} parameter of domain Ω_a etc.

$$p_{\omega_a}(\omega_a^1, \ldots, \omega_a^n) = \frac{\left[\int_{\Omega_b} d\omega_b \frac{f^2(\omega_a^1, \ldots, \omega_a^n, \omega_b^1, \ldots, \omega_b^n)}{p_{\omega_b}(\omega_b^1, \ldots, \omega_b^n)}\right]^{\frac{1}{2}}}{\int_{\Omega_a} d\omega_a \left[\int_{\Omega_b} d\omega_b \frac{f^2(\omega_a^1, \ldots, \omega_a^n, \omega_b^1, \ldots, \omega_b^n)}{p_{\omega_b}(\omega_b^1, \ldots, \omega_b^n)}\right]^{\frac{1}{2}}}. \tag{5.6}$$

The result for $p_{\omega_b}(\omega_b^1, \ldots, \omega_b^n)$ is similar. Thus the following is used to estimate the contribution of each bin in the case of importance sampling

$$d_i = \frac{K}{N} \sum_{x \in [x_i, x_i + \Delta x_i]} \Delta x_i^2 f^2, \tag{5.7}$$

where Δx_i is the width of partition i and f is the function to be integrated. For stratified sampling the variance of each bin is used this is calculated as

$$\sqrt{\left(\frac{\sum f}{N}\right)^2 - \frac{\sum f^2}{N}}. \tag{5.8}$$

Given these equations we can now generate a series of partitions that may be used to sample the function. The method for doing this depends on the criteria that we are using. For importance sampling we wish to make each box contribute the same amount and for stratified sampling each box should have the same variance. Stratified sampling is only practical with a small number of dimensions due to dimensional explosion. The algorithm shown in Figure 5.18 can be used to rearrange the size of the partitions so as to achieve either of these goals, by storing either integral estimates or variances in the bins.

Figure 5.19 shows the VEGAS algorithm used to integrate the function $\sin\theta$ using importance sampling. It used five iterations of the algorithm just described. Note that in the diagrams the width of the box represents the importance of a region. The height in this case is the estimated value of the function in that region generated after the region was subdivided. A perfect solution for importance sampling would have each box occupy an area of equal value, thus all boxes should be the same height. The noise present in naïve Monte Carlo integration is interfering with the construction of the bins hence the inaccurate results, represented by the small variations in the height of each bin. To avoid rapid destabilising changes to the PDF, a number of different approaches are used: A smoothing function is passed over the results for each bin, this function linearly interpolates the results thus averaging out the noise. This is generally

```
total = 0
FOR EACH bin
        total = total + $\text{bin}_i$.value
END FOR
$\Delta s = \frac{total}{\text{number of bins}}$
min = 0.0
$i = j = 0$
$s = 0.0$
DO
    $\text{tmin}_i$ = min
    WHILE $s < \Delta s$
            $s = s + \text{bin}_j$.value
            min = $\text{bin}_j$.min
            $d = \text{bin}_j$.value
            $\Delta d = \text{bin}_j.\text{max} - \text{bin}_j.\text{min}$
            $j++$
    END WHILE
    min = min + $(1.0 - \frac{s}{d})\,\Delta d$
    $\text{tmax}_i$ = min
    i++
WHILE $(i < n - 1)$
$\text{tmin}_i$ = min
$\text{tmax}_i = \text{bin}_{n-1}$.max
FOR EACH bin
        $\text{bin}_i.\text{min} = \text{tmin}_i$
        $\text{bin}_i.\text{max} = \text{tmax}_i$
END FOR
```

Figure 5.18: Rebinning Algorithm

good if there is a low number of samples. Figure 5.20 shows the results for $sin\theta$ using linear interpolation to smooth the function. Smoothing is not always enough therefore damping is used to control the amount of adaption. Two different methods can be used to do this. Firstly transition from the old to the new can be damped by limiting the amount of change possible, this allows a smoother more stable PDF to evolve (see Figure 5.21). We have a choice of two equations for this, the first being

$$(1 - r)d_i + rp_i = d_i, \tag{5.9}$$

where r is the ratio between the original value d_{i_1} and the proposed value p_i. This equation combines the new estimates with a set ratio of the previous estimate. This is only possible for the hypercube method. The second method is taken from the original VEGAS method, which uses the following formula to dampen the amount of change:

$$d_i = \left(\frac{1 - d_i}{\ln(1/d_i)}\right)^{\alpha}, \tag{5.10}$$

where d_i is the estimate for bin i and α is used to control the amount of change. A graph showing how this function varies with d_i and α is shown in Figure 5.23. Finally, by combining the damping with the smoothing, a better solution is obtained as shown in Figure 5.22.

Despite producing initial good effects, especially with small numbers of sam-

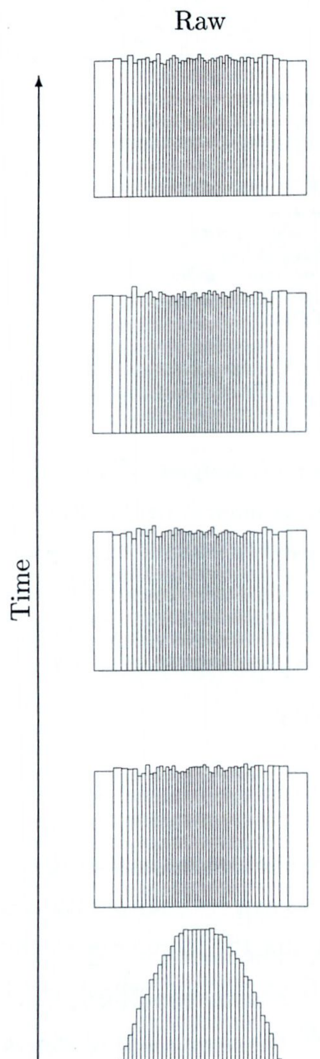


Figure 5.19: Series of PDF's by VEGAS algorithm for $sin\theta$ using raw data

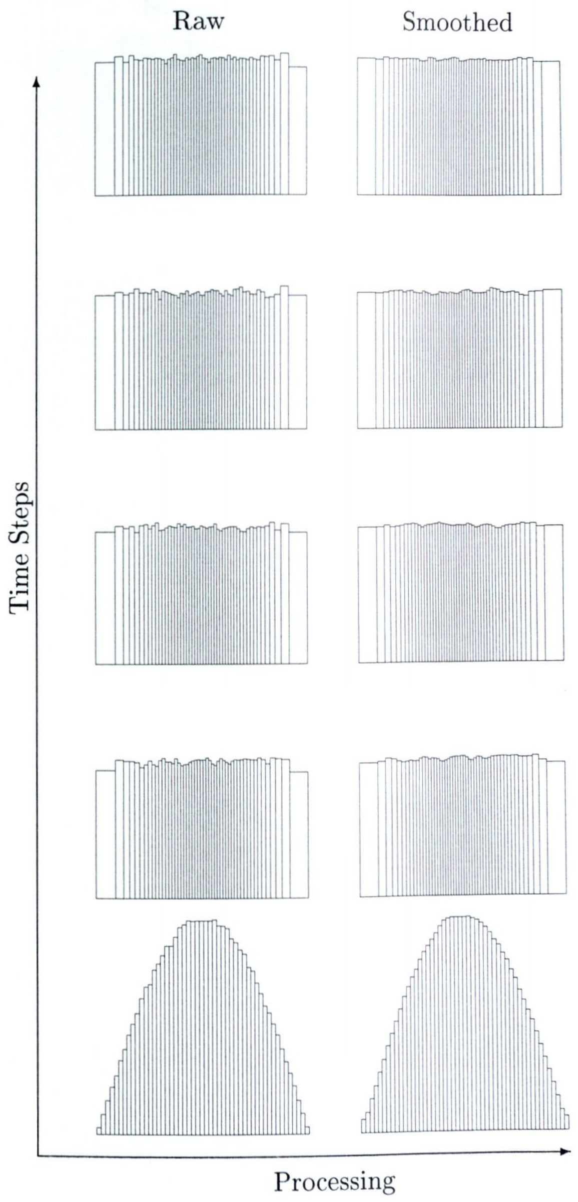


Figure 5.20: Series of PDF's by VEGAS algorithm for $sin\theta$ using smoothed data

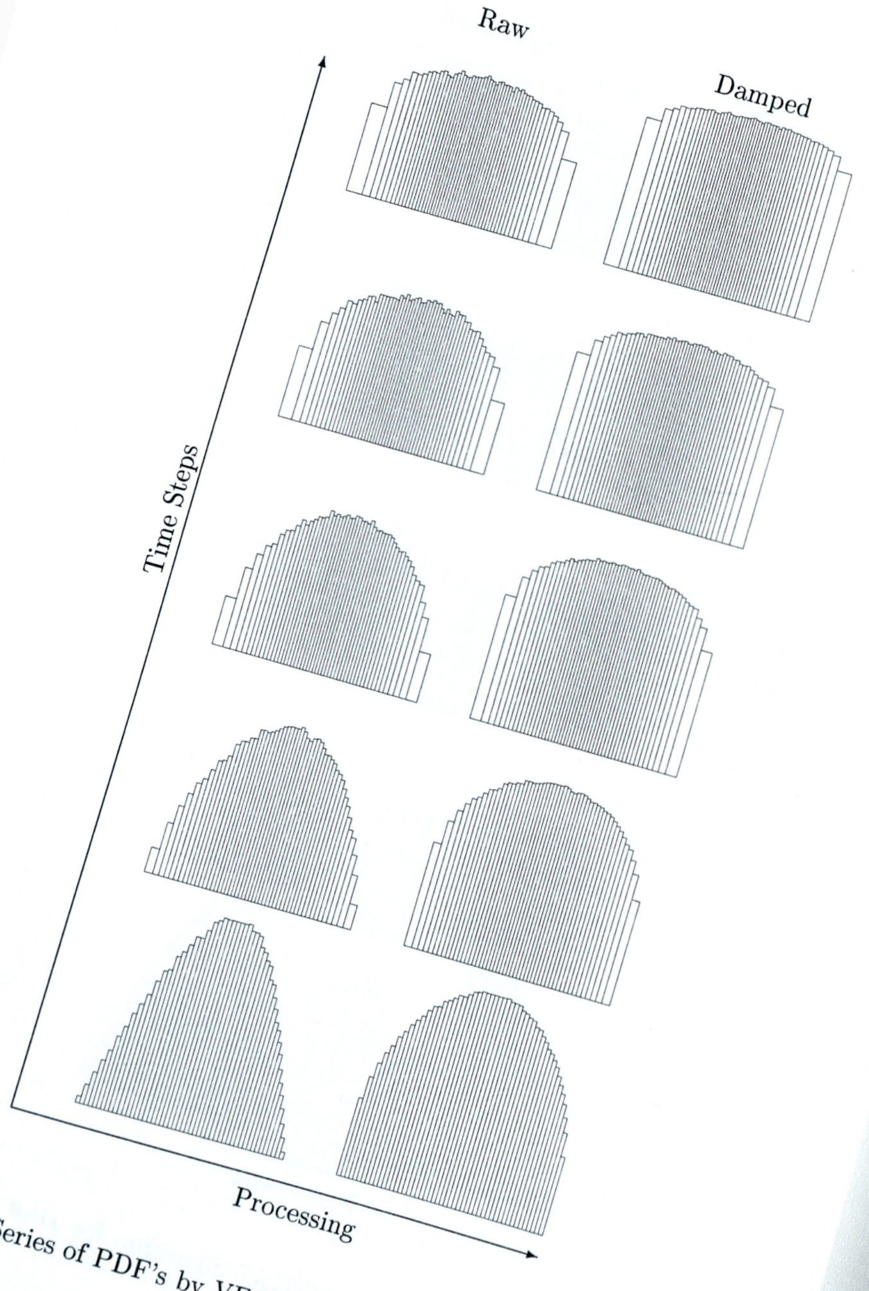


Figure 5.21: Series of PDF's by VEGAS algorithm for $sin\theta$ with damping

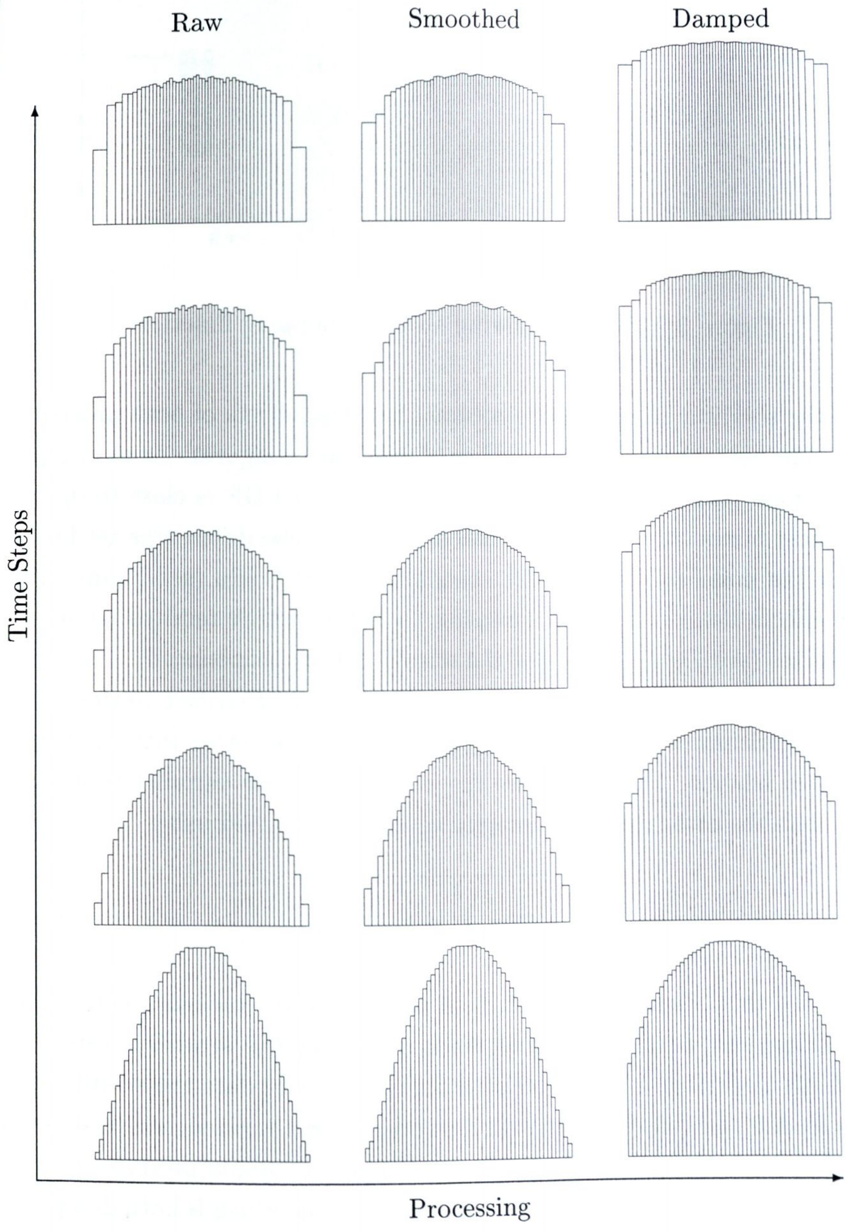


Figure 5.22: Series of PDF's by VEGAS algorithm for $sin\theta$ with damping and smoothing

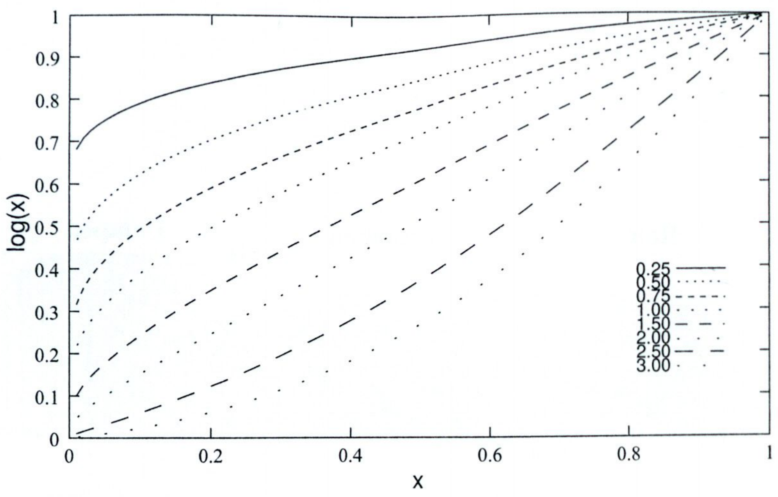


Figure 5.23: Graph showing VEGAS damping function

ples, as the solution becomes more accurate negative effects of both procedures appear. The smoothing allows a number of different configurations to masquerade as a good solution. This tends to happen as the PDF is close to optimal (where most boxes only differ by a small amount). These differences get filtered out with the noise and hence don't contribute to the boxes (see Figure 5.24). Similar effects happen with the damping algorithm which tend to ignore low valued boxes (see Figure 5.25), but the error is not as pronounced.

Damping is far more beneficial when used with the stratified version of the VEGAS algorithm, whereas smoothing appears to work better with the importance based scheme. This is because the estimation of the variance is far more chaotic than the estimate of the integral which is calculated using

$$\sqrt{\left(\frac{\sum f}{N}\right)^2 - \frac{\sum f^2}{N}}. \tag{5.11}$$

To illustrate this, Figures 5.26,5.27, 5.28 and 5.29 show various combinations of smoothing and damping on $sin\theta$ when used with the stratified version of the VEGAS algorithm. Notice how with the unprocessed data the stratification appears to shift radically between each iteration and in the smoothed version this effect is lessened somewhat. However, the dampened version produces a much more stable partitioning, the best version being the one which is both dampened and smoothed. Figures 5.30 and 5.31 illustrate the problems with smoothing and damping when large numbers of samples are used. Damping can cause the low valued areas to be underestimated. However, such excessive numbers of samples are almost never used in global illumination. If this problem does occur, the damping or smoothing can be switched off after a number of iterations to

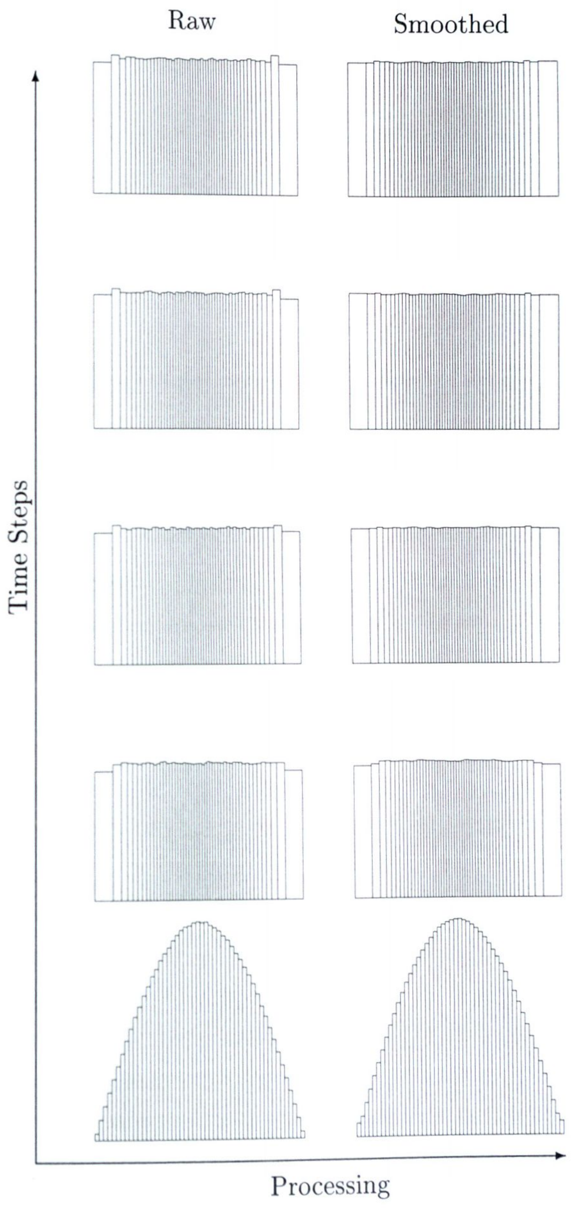


Figure 5.24: Graph showing errors caused by smoothing in VEGAS algorithm

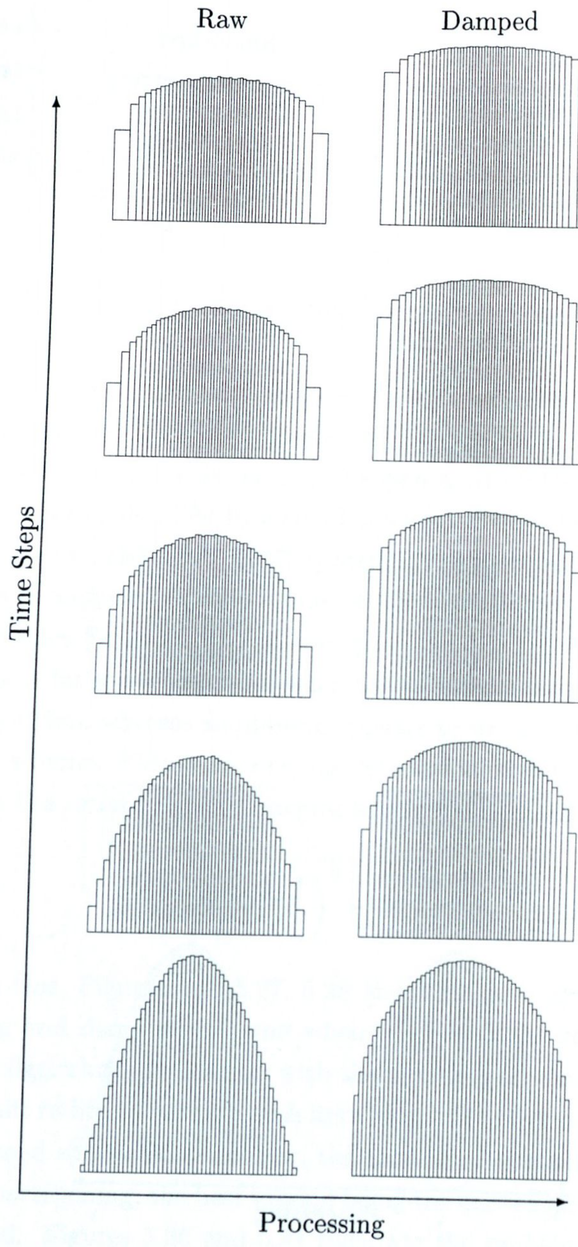


Figure 5.25: Graph showing errors caused by damping in VEGAS algorithm

attain better convergence. Unfortunately there is no way to recognise this in global illumination problems. Knowing the form of the importance function is the same as knowing function, in simpler integration problems the smoothness of the function could be used. This is not a good technique for global illumination as almost nothing is known about the function in question.

The key to the VEGAS algorithm lies in the combination of the results from each iteration. This is done by using the estimated variance to weight the contribution of each iteration. This is achieved as follows

$$a = \frac{\sum_{i=0}^{n} \frac{r_i}{v_i}}{\sum_{i=0}^{n} \frac{1}{v_i}} \tag{5.12}$$

where v_i is the variance of iteration i and r_i is the estimate for the integral, n the number of iterations and the overall result is given by a.

5.6 Stratified Integration Methods

Stratified sampling works by splitting up the domain of integration into sub-regions which are then integrated separately. This allows more samples to be taken in a sub-region where they are needed. Splitting is a form of partitioning which divides or splits ray paths. This is used to good effect with the direct lighting [119] method which splits a path to fire a ray at the light source. Another method of stratification commonly used in rendering systems is that of dividing BRDFs into a specular and diffuse part which are then solved separately. Unfortunately this technique requires knowledge of the BRDF or scene but has been successfully employed in the Radiance rendering system [134]. The two previous methods are based on a technique called splitting [5] where by ray paths are split.

The following new algorithms achieve stratification via a different approach, by breaking up the domain of interest into a number of different regions which are treated separately. This approach is novel in that rather than splitting ray path in two it divides the possible set of ray paths (see Figure 5.32). Thus partitioning defines the possible set of directions a ray path can travel, whereas splitting just breaks a single path into two directions which may or may not be the same. These techniques serve to reduce the overall variance of the estimate. However stratified sampling techniques can be used in another way, this is to facilitate parallel rendering. So rather than stratifying the problem to reduce the variance, other criteria which pose problems to parallel rendering such as communication overhead, data locality and ray coherence are used as stratification criteria. While this approach on its own does not directly improve the

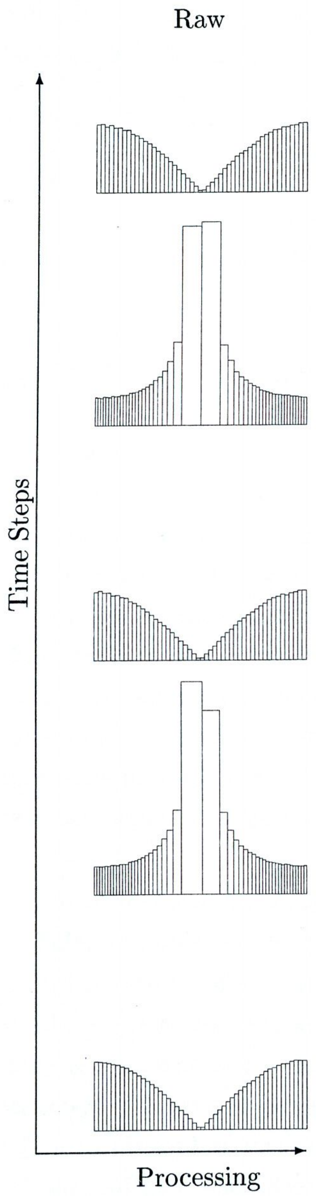


Figure 5.26: Stratified VEGAS algorithm without processing

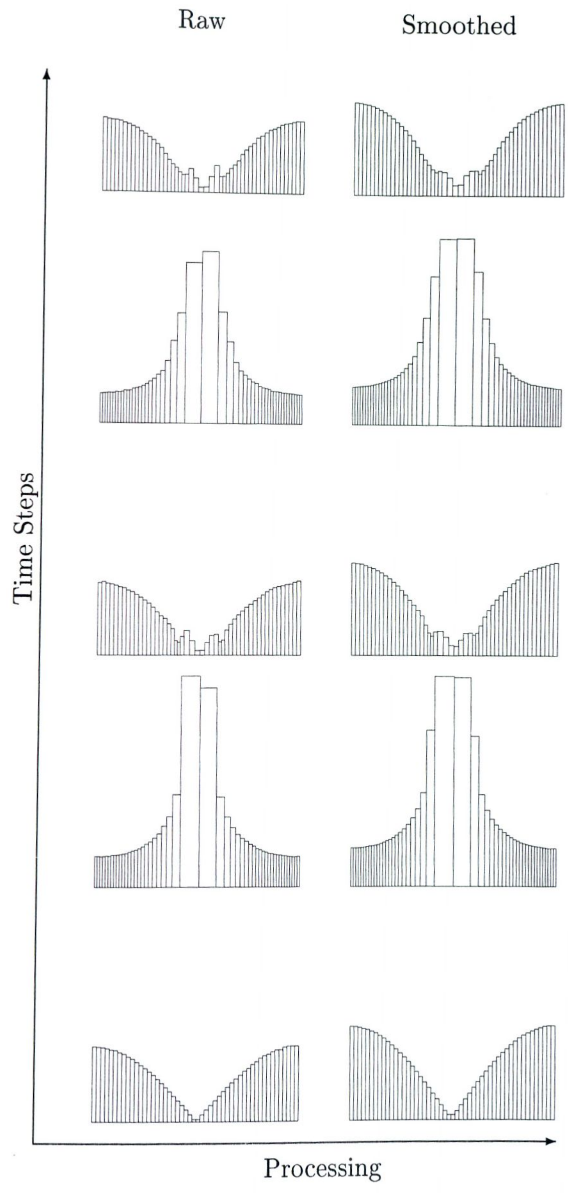


Figure 5.27: Stratified VEGAS algorithm with smoothing

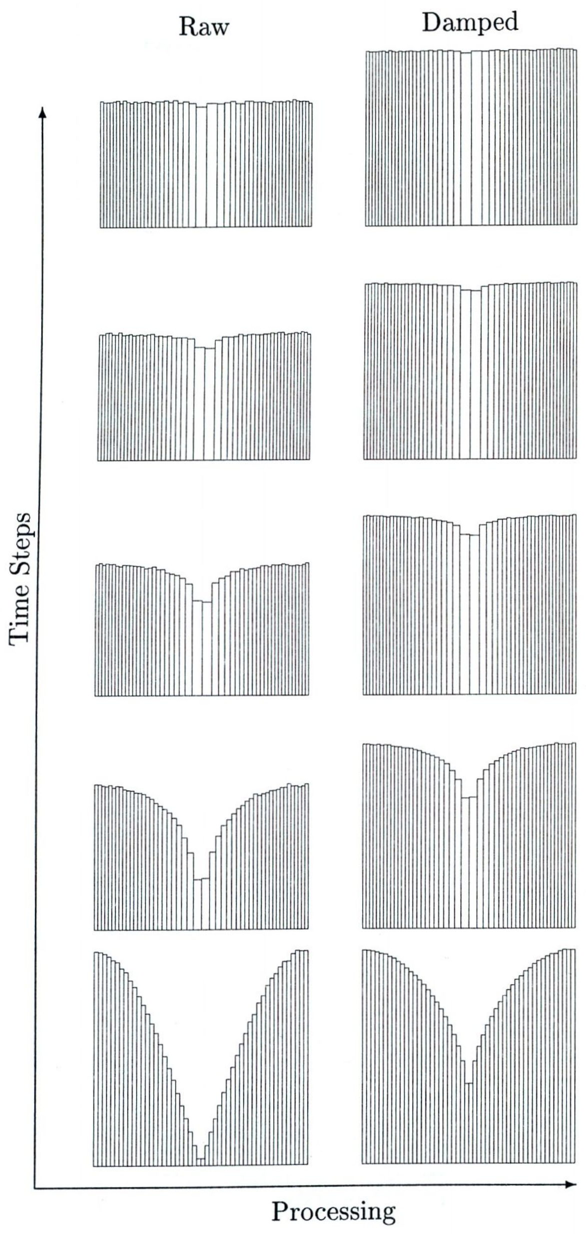


Figure 5.28: Stratified VEGAS algorithm with damping

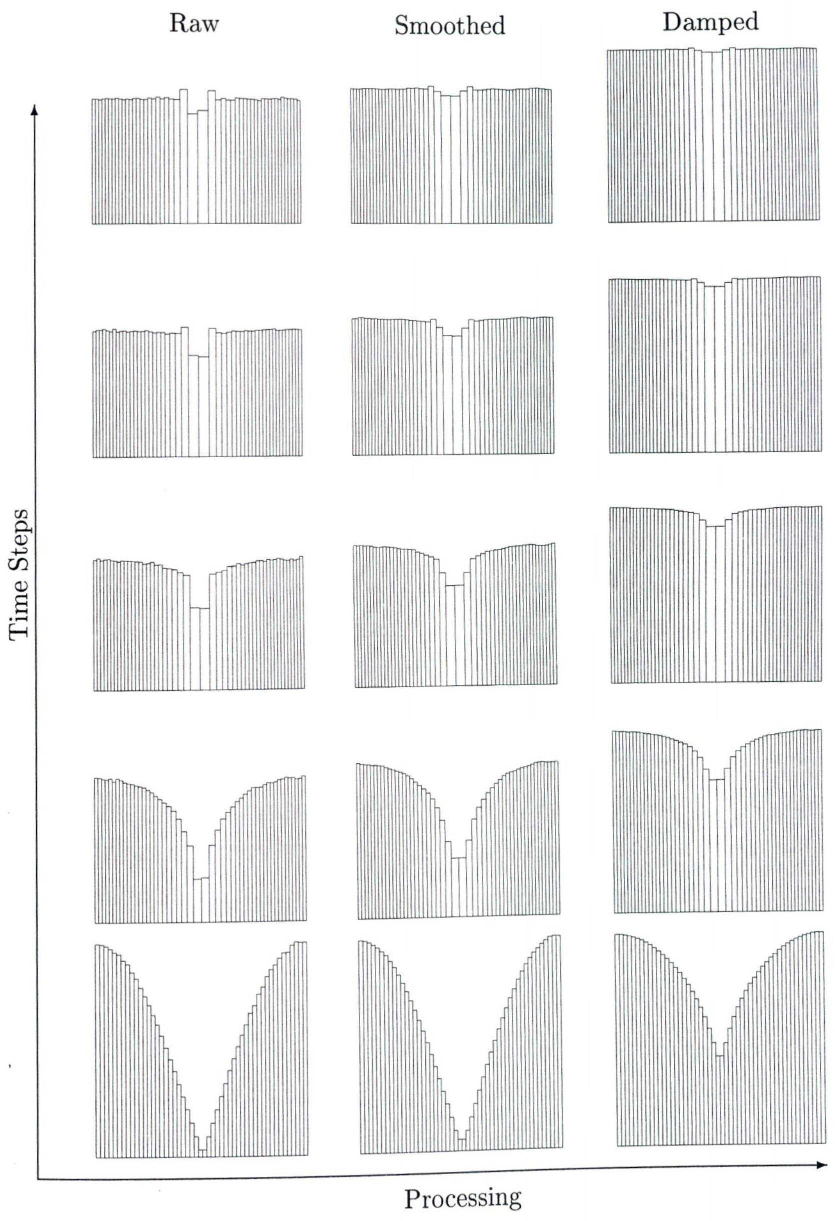


Figure 5.29: Stratified VEGAS algorithm with smoothing and damping

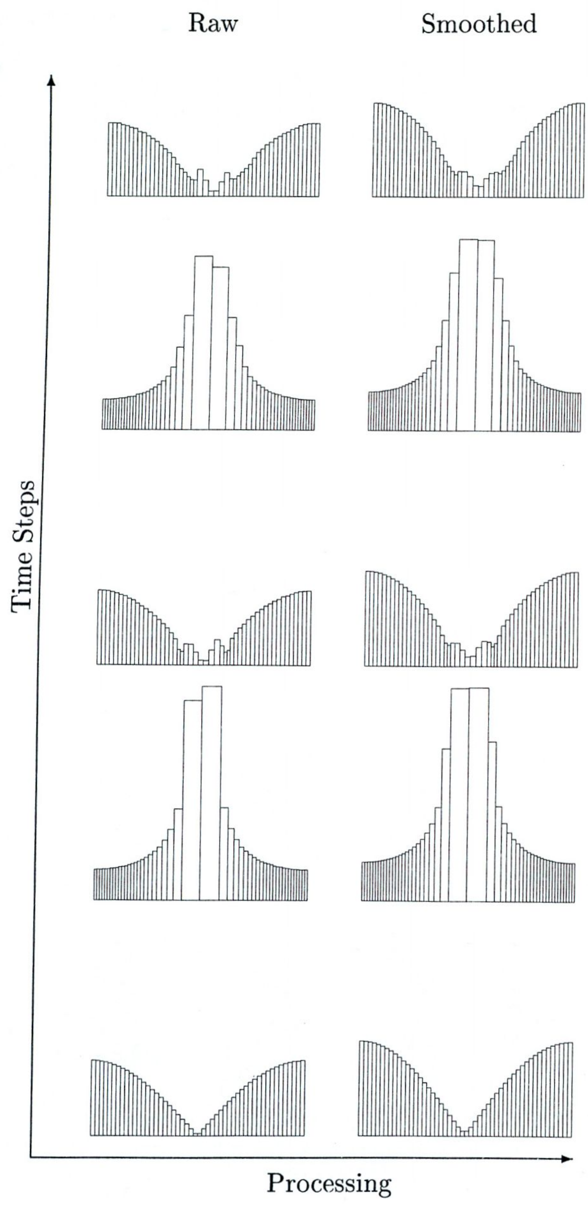


Figure 5.30: Stratified VEGAS algorithm errors due to smoothing

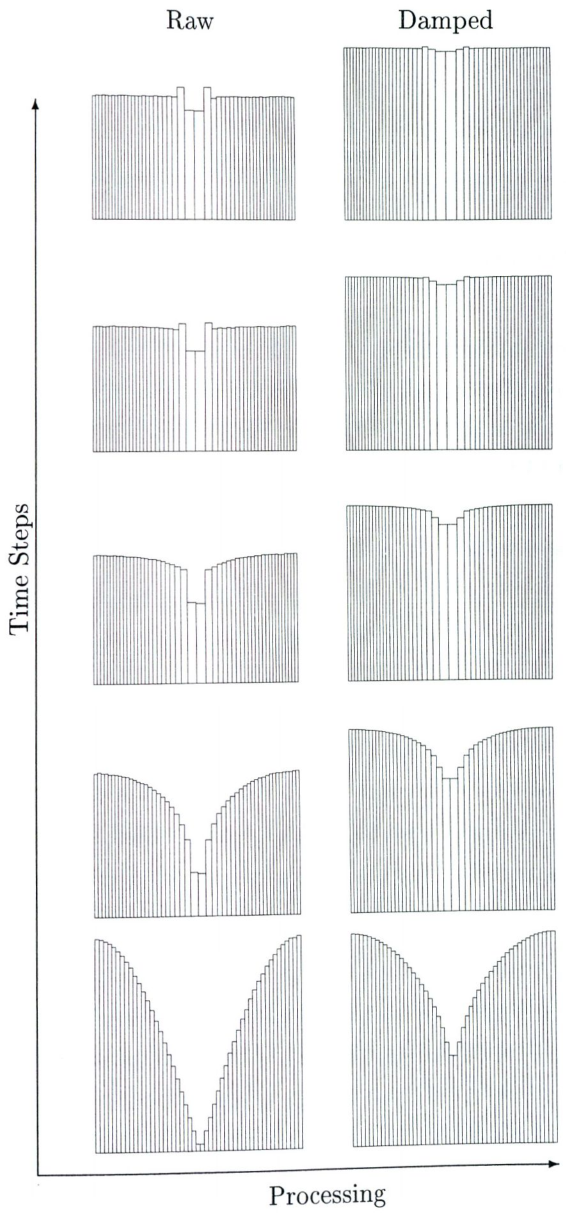
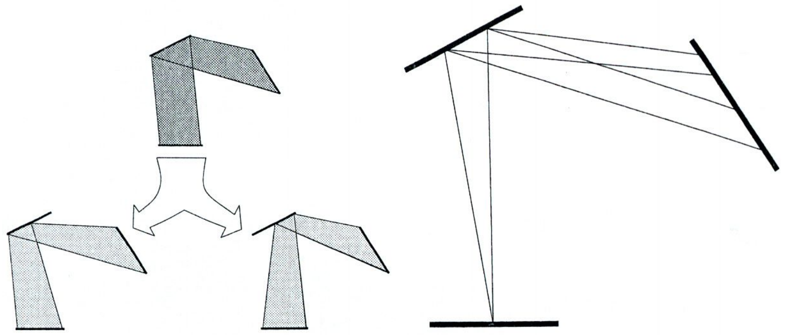


Figure 5.31: Stratified VEGAS algorithm error due to Damping



(a) Partitioning (b) Splitting

Figure 5.32: Partitioning a domain is not the same as Splitting a ray

convergence, it does improve the efficiency of the parallel technique, which in turn speeds up rate of convergence. Partitioning can thus be put to various uses. What follows is a discussion of some algorithms and metrics which guide these algorithms and the theory behind them.

5.6.1 Theory

In order to decide where to direct samples to estimate the integral of a function f, knowledge about the function's behaviour is needed. One way of doing this is to break it up into a series of sub-integrals:

$$\langle f\rangle' = \langle f\rangle_{\Omega_1} + \langle f\rangle_{\Omega_2}, \tag{5.13}$$

where $\langle f\rangle_{\Omega_1}$ is the estimate for the function over region Ω_1 and $\langle f\rangle_{\Omega_2}$ is the estimate for the function over region Ω_2. It can be easily shown that the variance of this is equal to

$$Var\langle f\rangle' = Var\langle f\rangle_{\Omega_1} + Var\langle f\rangle_{\Omega_2}. \tag{5.14}$$

Thus, to reduce the overall variance we need to reduce the variance of the estimate for region 1 or 2. Using equation 5.14 we can show that

$$Var\langle f\rangle' = Var\langle f\rangle_{\Omega_1} + \ldots + Var\langle f\rangle_{\Omega_n}, \tag{5.15}$$

It follows that to reduce the overall variance we have to reduce the variance of each sub-integral. The variance of the estimator $Var\langle f\rangle$, which measures the square of the error of the Monte Carlo Integration, is related to the true variance of the function by

$$Var\langle f\rangle = \frac{Var(f)}{N}, \tag{5.16}$$

where N is the total number of samples used, giving us the following

$$Var\langle f\rangle' = \frac{Var(f)_{\Omega_1}}{N_{\Omega_1}} + \ldots + \frac{Var(f)_{\Omega_n}}{N_{\Omega_n}}, \tag{5.17}$$

where N_{Ω_i} is the number of samples used in region i. Equation 5.15 is therefore minimised by

$$\frac{N_{\Omega_i}}{N} = \frac{Var(f)_{\Omega_i}}{\sum_{j=0}^{n} Var(f)_{\Omega_j}} \qquad \forall i. \tag{5.18}$$

Since standard Monte Carlo methods are not being used, equation 5.18 may not hold so instead the following is used:

$$\frac{N_{\Omega_i}}{N} = \frac{Var(f)_{\Omega_i}{}^{\frac{1}{1+\alpha}}}{\sum_{j=0}^{n} Var(f)_{\Omega_j}{}^{\frac{1}{1+\alpha}}} \qquad \forall i, \tag{5.19}$$

where α is an arbitrary scalar value. Empirical experiments have shown that when the equation is used to access sample opportunities recursively, $\alpha \approx 2$ gives the best estimates. Further details, including the derivation of the previous equations, can be found in [99, 100]. This, however, is not the prefect solution in global illumination as there can be significant time differences in evaluating various samples. Simply estimating the variance may not be a sufficient optimisation if the samples which improve the accuracy of the result take significantly longer to evaluate, thereby reducing the gains of using a better set of samples. In order to address these concerns the time taken to evaluate samples in a given region must be considered and worked into the equation. An example of where samples take longer is when we partition the path length parameter. This parameter specifies the length of the path thus requiring more time to evaluate for bigger values i.e longer paths.

5.6.2 Stratified Sampling Algorithm

The Miser [100] algorithm is based on a technique called recursive stratified sampling [99]. This version of the Miser algorithm is essentially a hybrid between the original VEGAS algorithm and the original Miser algorithm, thereby gaining the benefits of both approaches. It differs in seven ways from the original by providing:

- new metrics to estimate the variance,
- a weight for each dimension,
- specification of an initial level of sub-division,
- dithered splitting of dimensions,

- an arbitrary number of divisions per level,
- estimate reuse if necessary (similar to VEGAS method),
- sensible partitioning of dimensions based on VEGAS technique or original Miser algorithm.

A number of different metrics can be used to estimate the variance. These are the *greatest difference* metric, *maximum value* metric and Monte Carlo estimation. The greatest difference metric uses the difference of the squares of the maximum and minimum values for a region rather than the actual estimate. This is given by

$$Var\langle f\rangle_\Omega = \left[max(f(x_i)) - min(f(x_i))\right]^2, \tag{5.20}$$

where x_i is one of the N samples allocated to determine the variance. This estimate is biased as the number of samples is increased, but since there should be an approximately equal number of samples in either of the sub-regions being tested, this is acceptable.

The maximum value metric is based on the greatest difference metric (and hence is also biased), but we assume the lowest value is always zero so we only need the highest. This metric is given by

$$Var\langle f\rangle_\Omega = \left[max(f(x_i))\right]^2. \tag{5.21}$$

Lastly, Monte Carlo estimation of the second moment $Var\langle f\rangle_\Omega$ can be used, which is calculated using

$$Var\langle f\rangle_\Omega = \frac{1}{N}\sum_{i=0}^{N} x_i^2 - \left[\frac{1}{N}\sum_{i=0}^{N} x_i\right]^2. \tag{5.22}$$

This gets increasingly inaccurate as the number of samples drops, and suffers from the slow convergence associated with Monte Carlo methods.

Since the algorithm is using samples to determine a suitable stratification, it makes sense to use this information not just to determine which dimension to split, but also to choose where to split the chosen dimension, or even whether to split it in more than one place. This allows as much information to be gained from samples as possible. For instance, usually regions of high variance do not just occur in one area (e.g. two light sources in a scene would cause spikes in two different solid angles). Also, in global illumination areas of high variance tend to be in small regions of the domain, so splitting the domain in half does not effectively partition the regions of high variance. To facilitate these requirements our algorithm uses a series of n bins for each dimension. The

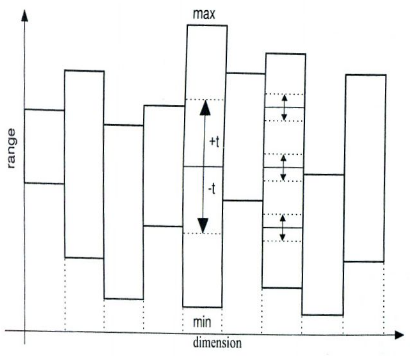


Figure 5.33: Dithering divisions between bins

variance and estimate for each bin of each dimension is determined. When a dimension is chosen an approximation of how the variance is distributed across that dimension will have been determined. Thus the algorithm now has an indication as to where it should split the domain and into how many parts. This is achieved using the partitioning scheme from the VEGAS algorithm, the variance being used as the subdivision criteria.

In order to provide a more adept solution to the rendering equation, the user is able to apply a weighting for each dimension. This helps the stratification algorithm to choose which dimension to split, especially when it has to pick one randomly or when the variances do not differ by much. For instance, it is generally better to split dimensions which correspond to primary rays since they tend to affect the result more than parameters which alter rays on later bounces. The weights allow us to indicate the importance of each dimension to the stratification algorithm.

Dithering can be used to divide the region into unequal sized sub-regions rather than evenly spaced ones, this reduces the correlation caused by the uniform subdivision scheme. This is achieved by shifting the dividing line between two bins left or right by a fraction of the overall width of the bin using a random value (see Figure 5.33). This technique is only used when the original Miser binning scheme is used as the VEGAS algorithms approach generally proves more effective due to its adaptability.

Due to the fact that in rendering we tend to use relatively low numbers of samples because of the cost of evaluating the rendering equation, we cannot always afford to throw away samples. To help improve our estimates, the algorithm has the ability to reuse the samples used in estimating the variance, to provide another estimate for the integral. This tends to produce biased results. However, with low numbers of samples the results obtained give a better overall estimate. These two estimates are combined by the ratio of their variances,

```
FUNCTION integrate(domain,samples)
    IF not enough samples
        integrate normally
    ELSE IF depth < initial sub-division depth
        choose dimension to split according to weights
        split domain of integration along chosen dimension into n bins, dithering if requested
        allocate samples/bins to each bin
        FOR each bin i
            integrate(bin i, bin i samples)
        END FOR
    ELSE
        estimate variance in each bin
        choose dimension to split according to weights and variance
        split domain of integration along chosen dimension into n bins, dithering if requested
        allocate samples to each bin according to variance
        FOR each bin i
            integrate(bin i, bin i samples)
        END FOR
    END IF
END FUNCTION
```

Figure 5.34: Stratification Algorithm

similar to the method used in the VEGAS algorithm.

The final alteration is to provide an initial level of subdivision before any adaptive subdivision starts, like jitter sampling in n dimensions, this reduces the effect of badly estimated variances. Unfortunately, as a side effect it will also reduce the gains caused by good estimates, because the samples are initially spread evenly throughout the domain of integration.

The algorithm works as follows: Given an initial number of points N, use a portion of these to estimate the variance across the function being integrated. The estimate of the variance is obtained for each dimension. Pick the dimension with the largest overall variance and divide it into n sub-regions using the information in the bins. Repeat the algorithm in these new regions until a specified number of divisions has occurred or there are not enough samples to continue, in which case integrate normally over these regions (see Figure 5.34).

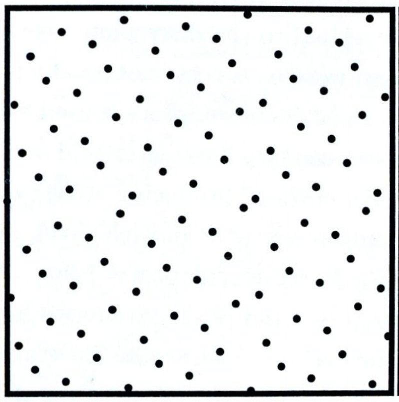
5.6.3 Sample Sequences

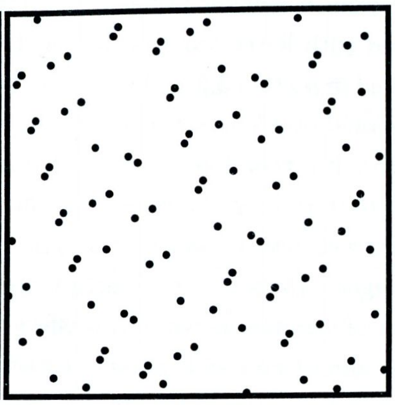
To improve the convergence of Monte Carlo integration, random and quasi-random sequences are constructed based on linear congruential and radical inverse methods. The quasi random sequences help to improve the convergence because they, by their nature, tend to spread their sample set over a given domain.

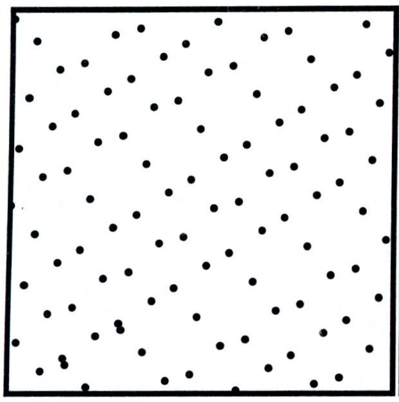
Due to the way that some quasi-random sequences are generated (such as the Halton and Hammersly sets), the higher order dimensions will have a greater jump between samples than lower order dimensions. This will affect the error, due to the fact that the bigger the prime used to generate the sequence, the larger the jump will be. The effect can be minimised by mapping parameters which contribute more, to a dimension which is sampled with a sequence generated from a smaller prime. The improvement is due to the better spread achieved with lower order primes, and is related to the discrepancy which was described in section 3.2.1. This is why quasi-random sets are not good with high dimensional problems especially when low numbers of samples are used (see Figure 5.35). It is therefore recommended that secondary bounces should use higher primes than primary bounces. This has the effect of producing strong correlation between pixels, because the same samples are used in each pixel, making the images look bad even though the error in the calculation is lower. As the number of samples increase, this effect decreases and produces images in which there is almost no visible noise or correlation effects, Figure 5.36 illustrates this, an edge-enhancement algorithm is used to highlight the effect. Scrambling of the quasi-random samples can be used to trade the correlation for noise. The quasi-random number sequences also have deterministic error bounds which are unfortunately tied to the dimension of the function being integrated, these are better in nearly all cases than the standard random sampling strategies according to the discrepancy associated with each.

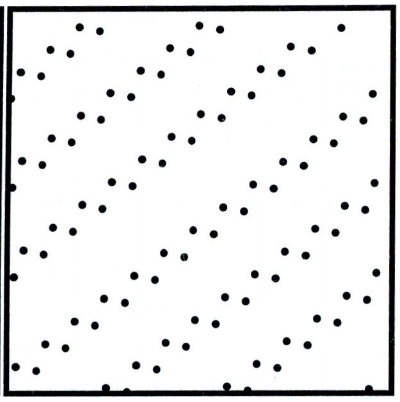
5.6.4 Stratification as a method for Parallel Computation

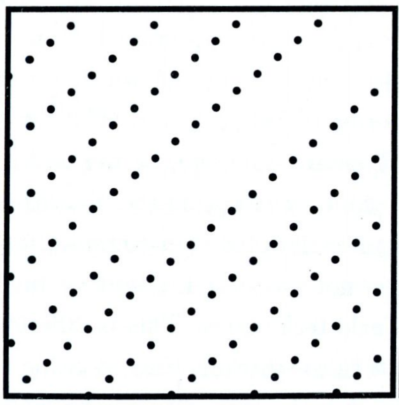
To date there have been many parallel approaches for ray based global illumination methods. These methods have adopted a demand driven, data driven or a hybrid scheme. Most approaches have looked at parallelising the process at the image level or the scene traversal level. Rarely has the scheduling of tasks resulted in a new algorithm being created or significant alterations to an existing one to make the program more effective in parallel. The greatest problem with creating a parallel ray tracing method is in finding and exploiting any coherency that a given algorithm may contain, and this is very important for data driven methods. Techniques such as breath first ray tracing [96] which attempt to re-order the access to the scene and buffering techniques which wait for enough queries to warrant further processing [96] create coherence by reordering work. These methods have typically not been done by changing the algorithm itself but by caching and buffering requests and or data for later computation. This algorithm unlike previous approaches does not try to find coherency, but rather attempts to create it by using Monte Carlo techniques. This is different from other approaches [96, 105] which attempt to re-schedule tasks by using caching techniques. By partitioning the sampling domain, ray paths can be made more coherent because this forces samples or ray paths to pass through similar regions of the scene. This means that rays now travel in "shafts" or "bundles" and techniques such as pyramid clipping [127] and shaft culling [50] can be used to exploit this, previously such methods were only possible on primary rays.


(a) Dims 0,1


(b) Dims 2,3


(c) Dims 4,5


(d) Dims 6,7


(e) Dims 8,9

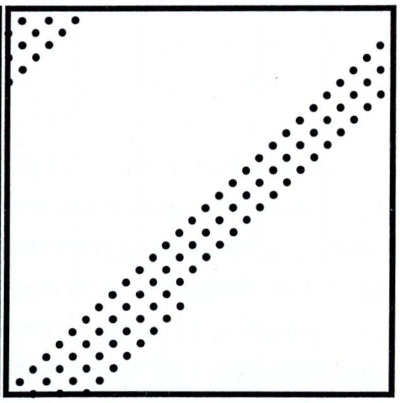

(f) Dims 10,11

Figure 5.35: Two dimensional plot of successive dimensions of a 100 sample quasi-random sequence

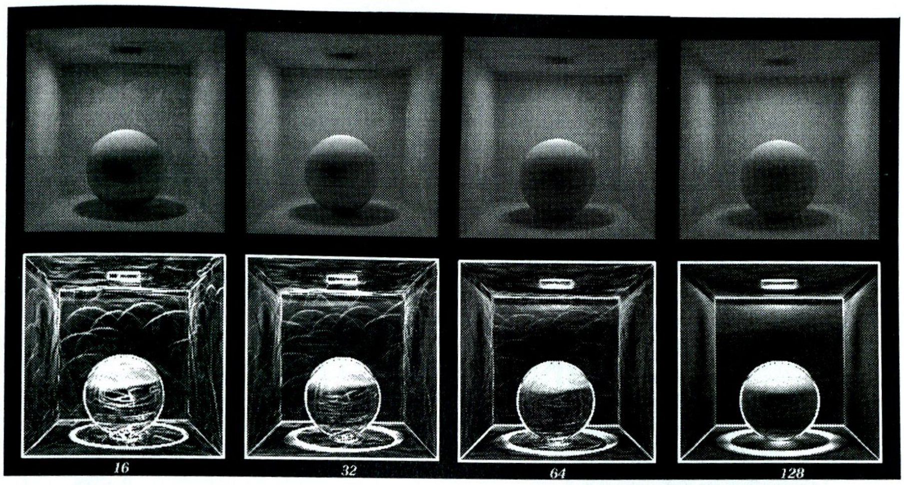


Figure 5.36: Correlation effects due to quasi-random sampling with varying numbers of samples

Since a smaller subset of the scene is now being queried rather than the random access of random sampling, processor caches and virtual memory become far more effective as these partitions can be limited to areas of the scene which fit in real memory thereby reducing the amount of paging. Thus for small scenes super-linear speedup may be achievable given a sufficient processor cache size.

Using Stratified Sampling

This algorithm is based on Monte Carlo path tracing and uses the idea of integrating over a section of the domain of integration. By splitting up the domain of integration, ray paths can be focused on set regions of the scene. This enables it to sample over a smaller region of the scene, which for instance would fit in main memory. When finished sampling this region it can then alter the domain and repeat the sampling process over a new region. This would typically cause new areas of memory to be paged into active memory and others to be paged out. By minimising this paging, the efficiency of the algorithm will be increased. The technique used allows the parallel scheduling algorithm to be removed from within the global illumination computations and a high level approach to be adopted. It is high level as it does not deal with details of rendering such as ray queries or BRDF evaluation but instead the solution of the whole equation. This is more modular and can thus be applied to many more methods since it controls the parallel computation from a high level rather than by using a low level method which is entirely hidden from the rest of the program. There is however a price to be paid because of the high level of this approach, as fine level of control and optimisation is not always possible as with

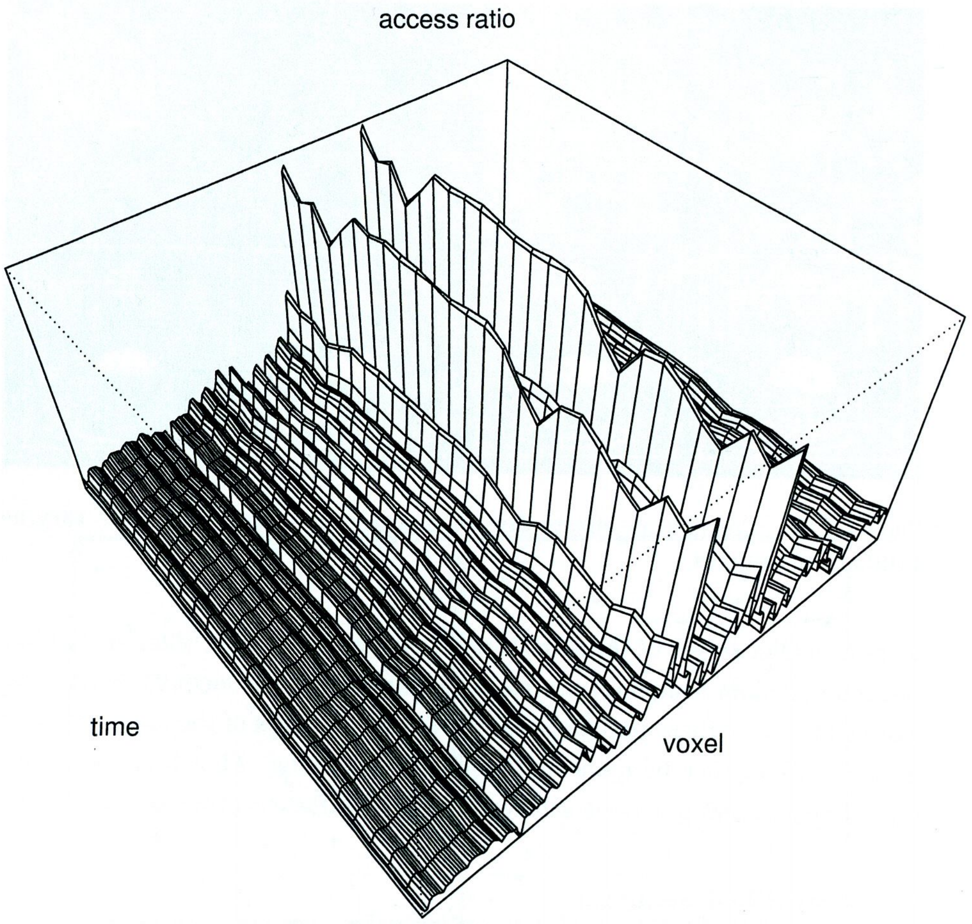


Figure 5.37: Voxel Access vs Time for Random Access

lower level approaches. Figure 5.37 shows the percentage of accesses over time, and Figure 5.38 is a graph which displays the access to the voxels of a scene using a completely random Monte Carlo process. Figure 5.39 shows the voxels of the scene where their size represents the ratio between the number of times they were accessed and the voxel with the most accesses.

From examining the previous graphs, it can be clearly seen that the voxel access is spread quite evenly throughout the scene, with the exception of areas around the light source or camera. These areas have a far higher number of accesses than anywhere else. This is because all ray paths must at least start in those voxels. When a naïve stratification scheme is applied to the standard particle tracing approach just used, the coherency of rays over time can be changed significantly. By breaking up the directions rays are permitted to travel in and where they are allowed to start, shafts or beams of ray paths are created. These beams increase the coherency of rays since many of them are now travelling

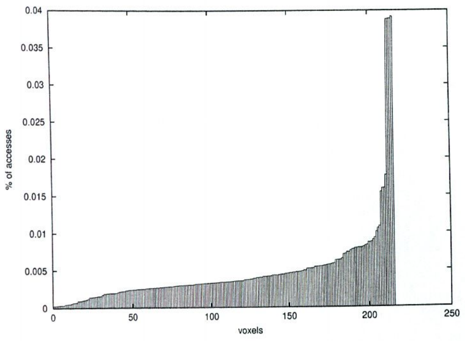


Figure 5.38: Voxel vs Percentage Accesses for Random Access

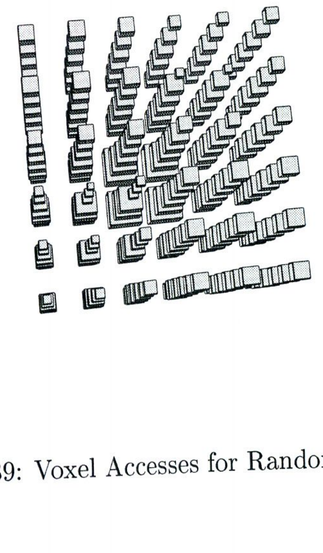


Figure 5.39: Voxel Accesses for Random Access

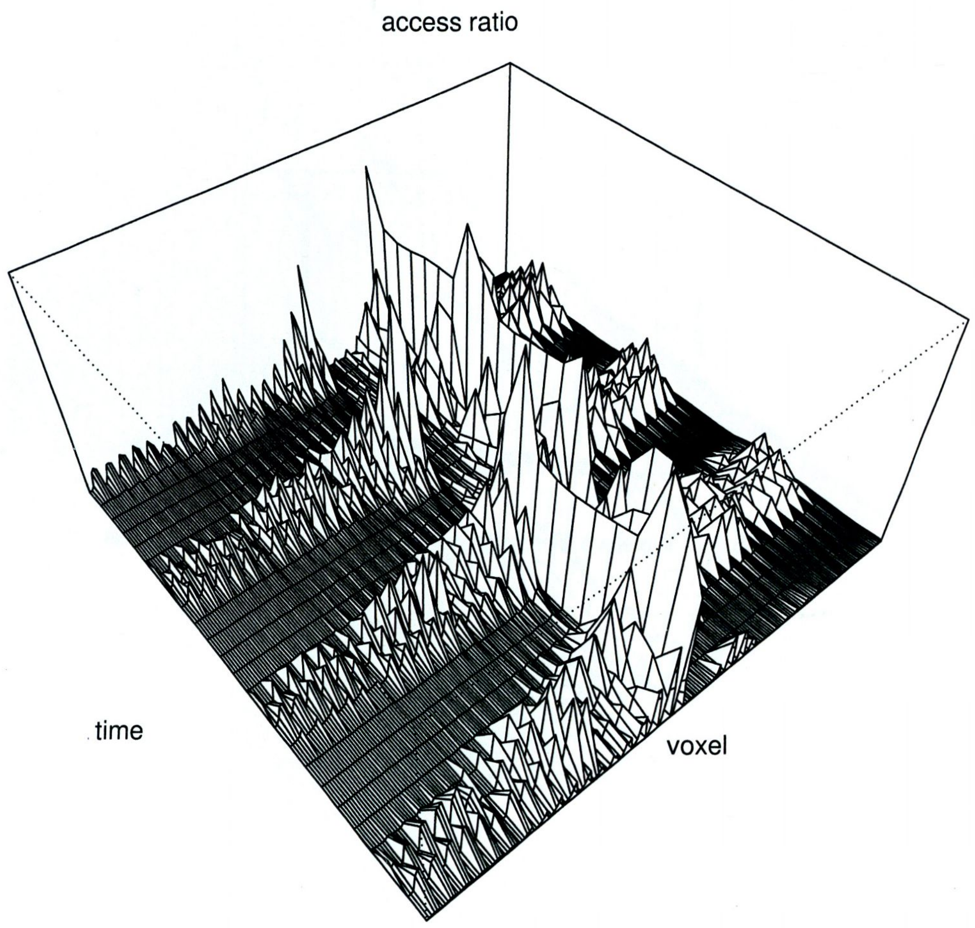


Figure 5.40: Voxel Access vs Time for Solid Angle Partitioning

in similar directions and intersecting with the same objects. This coherency is shown in Figure 5.40. Examining the overall number of accesses between the two methods shows that approximately the same number of accesses were made to each voxel with perhaps a slightly more uniform distribution caused by the partitioning scheme (see Figure 5.41) due to the uniform like sampling caused by partitioning. The differences between these distributions are quantified by Mitchell [82].

Applying this partitioning scheme with path tracing is almost identical to applying it to particle tracing. In fact the camera and film just provide an automatic initial partitioning and hence already possess a high level of coherency so partitioning is probably unnecessary for the primary ray. Pyramid clipping has been used to exploit this [127]. Subsequent rays however, need a high level of partitioning but unfortunately less samples are available so the number of possible partitions that can be used is reduced. Even small numbers of partitions

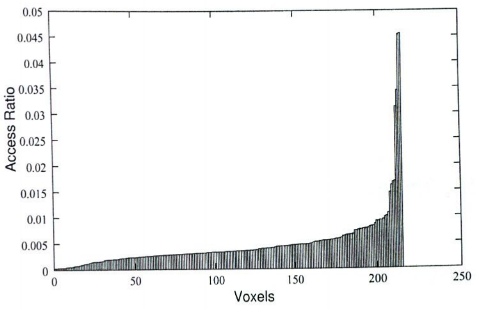


Figure 5.41: Voxel Access vs Percentage Accesses for Solid Angle Partitioning

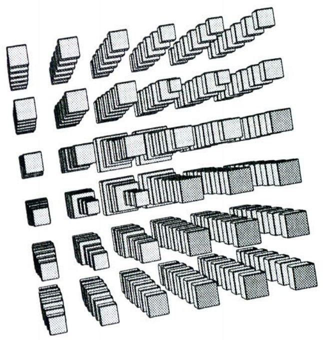


Figure 5.42: Voxel Access vs Time for Random Access

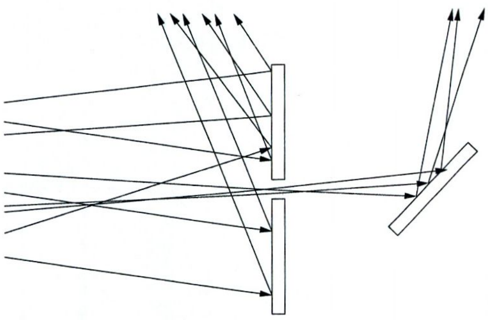


Figure 5.43: Beam split due to ray hitting different surfaces

will produce an improvement in coherency especially if the scene is largely flat or uses polygons. This is because rays coming from the camera are already partitioned into the pryamids caused by pixels any further partitioning futher constrins these ray thereby increasing partitioning.

Parameterising the Ray Path for Stratification

Unfortunately, using naïve stratification of the ray direction will not work for all scenes, due to a number of factors. The direction of the samples is dependent on the normal at the point of intersection and thus on the shape of the surface itself (see Figure 5.44). This causes beams to either converge to narrower beams, which is good for coherency, or to diverge and spread out which reduces coherency. If the rays in these beams strike different objects they may be scattered in completely different directions (see Figure 5.43) once again reducing coherency. In order to reduce this, enabling a sensible splitting of the domain of integration for our algorithm, axis aligned bounding boxes are used to parameterise the ray paths instead of the ray direction. These could be the voxels of a Spatially Enumerated Auxiliary Data Structure (SEADS) grid [38] or the bounding boxes [112] of clusters of objects in our scene. This provides us with a way of intelligently parameterising the domain of integration. Thus, by selecting groups of axis aligned boxes which are then sampled (see Figure 5.45), a group of paths can be specified. Each of these boxes is parameterised by two variables (u, v). The order of the boxes specifies the order in which rays will hit the objects. If the ray does not hit anything in the box it simply returns zero, thus maintaining coherency. Another possible parameterisation is to use a unified polar coordinate system which is very simple to use, but makes sampling BRDFs more difficult.

One of the problems with using classical ray tracing methods (i.e. distributed ray tracing [24, 118, 133]) is that many rays must be shot at each light source to estimate the light incoming to a point. These rays will be fired into the scene,

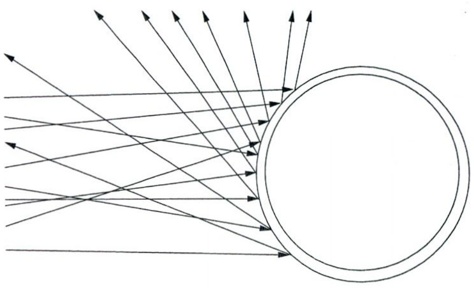


Figure 5.44: Beam diverges due to ray hitting curved surface

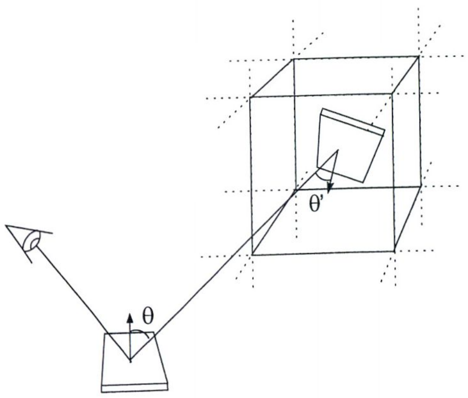
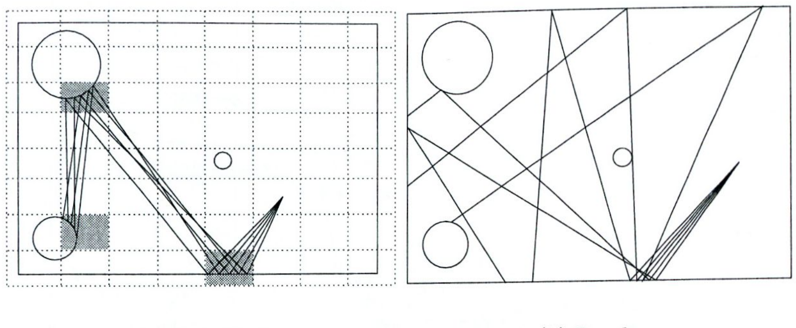


Figure 5.45: Sampling An Axis Aligned Bounding Box



(a) Stratified (b) Random

Figure 5.46: Stratification using Bounding Boxes vs Random Ray Directions

possibly accessing non-local memory. The path tracing methods have introduced ways to minimise this, by firing a single shadow ray to a randomly chosen light source on each bounce. This still causes a split in the ray path which causes random data to be required. However, the bi-directional path tracing method provides a potential solution to this, where each ray path ends on a light source thus reducing the need for shadow rays entirely. Another approach is to use the single path evaluation methods used with the previously described adaptive schemes. Ray paths which contribute to the illumination of a point on the film can now easily be described. By partitioning this parameterisation, which objects or parts of the objects are accessed can be effectively controlled. The virtual memory system of the host operating system will be able to effectively manage which region of the scene needs to be in memory. Thus there is no need to transmit any scene information to each computer, since each will load it into virtual address space by using memory mapped files which can store very large amounts of data (typically around 4Gb for PC's and most 32bit computers) as if they were in main memory. A scheme whereby scene information is transmitted when necessary could be used but this was not attempted.

The key to how the parallel method works is in the conversion of the parameter domain into a series of sub domains which can be solved later. To do this each dimension i is broken into a number of segments n_i, this then creates a total of $\Pi_{i=0}^{m} n_i$ sub-domains which can be distributed among the available processing elements. The simplest way to iterate through these is simply to calculate the total number of sub-domains and assign a given number to each one. An ordered string of these numbers represents a entire partition. This string can be converted to a single number by multiplying each number in the ordered string by its offset, similar to converting a number between different bases, except now each digit has a different base. This can be done using

$$\begin{aligned} v_0 &= s_i \\ v_n &= \sum_{i=0}^{n} s_i v_i \end{aligned} \tag{5.23}$$

where v_n is the integer representing the ordered string of length n and s_i the integer representing the segment of the i^{th} grouping. These numbers can then simply be converted back into a sub-domain by the following

$$s_i = \frac{v}{\Pi_{j=0}^{i-1} n_j} - int\left(\frac{v}{\Pi_{j=0}^{i} n_j} n_j\right) \tag{5.24}$$

which gives a segment number s_i for each parameter or dimension i of the domain of interest. This can easily be converted to the actual sub-domain for processing. The scheme reduces the amount of information that has to be sent between processing elements, in this case just two integer values for the range of sub-domains, allowing a small work queue size that can be efficiently managed, so reducing the time wasted during rescheduling.

The scheduling is separated from the partitioning and only deals with the integer ranges described. This allows for a variety of scheduling algorithms to be used. Currently a mix of static and work stealing algorithms as well as a simple demand driven scheme have been implemented. The work stealing initially sends every PE an equal portion of the work. Then later, when a PE runs out of work, it signals all the others PEs to send their task list, allowing the largest queue to be split in the ratio of the number of elements processed between it and the PE which has no work left. If any other PEs have no work, this procedure is repeated. While this is not a perfect load balancing algorithm, it does not suffer too badly due to the fact that rendering speed depends not only on the processor, but also on the scene. This scheme keeps the communication to a minimum. The static scheme uses the estimated power of each PE to estimate the proportion of the scene it should receive. Using a cost prediction algorithm [109] in conjunction with this may improve performance, but this has not been attempted. Many other schemes such as the diffusive methods [25] or some work stealing schemes may prove better at scheduling the work load.

View Dependent approach

The final problem that must be dealt with is how to split our domain. If the axis aligned bounding boxes are used then various combinations of n sets of boxes can be selected. If the boxes are too big, this causes large areas of the scene to be used due to the solid angle subtended. To avoid this a SEADS (Spatially Enumerated Auxiliary Data Structure) grid [38] is used which creates boxes of uniform size thus reducing the spread of rays through the scene (see Figure 5.46). Accordingly the form of the rendering equation used ends up as follows:

$$L_i(x) = L_e(x) + \int_S \rho(\omega_i, x, \omega_o) L_i(x) \frac{V(x,x') cos\theta' cos\theta}{||x - x'||^2} dA, \tag{5.25}$$

Where L_i is the radiance incoming to x, S is the set of all occupied voxels, ρ is the bidirectional scattering distribution function (BSDF) with ω being the i incoming or o outgoing direction, θ the angle between the surface normal and the incoming or outgoing ray, x and x' are points on surfaces and $V(x, x')$ expresses the visibility between the two points. This is essentially the surface to surface form of the rendering equation which uses a rejection sampling scheme to sample objects. Any empty voxels can be ignored, reducing the set of partitions that need to be processed. This is an important optimisation because the more voxels involved the more partitions needed such that there are n^d combinations. Thus the user must be wary of combinational explosion, a common problem with stratified sampling schemes.

Each PE is assigned a region of the image and a group of voxels from the boss. This is represented by two integers which are sent to the PE. The results are returned after all processing has been completed, thereby avoiding the extra communication costs of transmitting the results immediately. The extra cost incurred by transmitting the results immediately are because more than one partition may be assigned to a single pixel.

View Independent approach

The partitioning scheme described previously is not only applicable to integration based view independent schemes, but also to particle tracing methods. Since particle paths are described by a series of parameters, they can be split using the partitioning method. The modularity and flexibility of the framework allows the stratification technique to be combined easily with particle tracing methods. These methods work very well with stratification especially since the particle paths do not have to be joined to the eye as in the view-dependent approaches. A far greater number of samples are available, so more dimensions can be stratified with finer divisions. The view independent approach differs mainly in the use of the potential equation rather than the rendering equation used in the previous method. This allows the break up of the entire parameter space to proceed as before, using the same scheduling algorithm to control the work load. The images this time are not the final picture but the illumination maps or photon maps used to describe the illumination across the surfaces of the scene. The particle tracing approach would seem to favour a data distribution approach. Due to the volume of particle data generated (in excess of 100MB usually, see Figure 5.51), while the program is running in relation to the scene size it seems better to migrate the scene to the task rather than the other way around. Even small scenes require large volumes of particle data. Transmitting the data leads to increased costs as the particles used generally carry more information than their counterparts used to reconstruct the illumination on the surfaces. This is because once a particle has been stored on a surface, all unnecessary information is stripped away. Thus the stratification approach is used to constrain the particle paths to smaller areas of the scene. This enables a demand driven approach which has some of the properties of a data driven scheme. A number of issues arise regarding the density information that has been stored. If this data has to be moved with the scene this will cause problems for, as previously noted, this data tends to be larger than the size of the actual scene. Thankfully this is not necessary for the density representation schemes used, which are the illumination map and the photon map. Data from these two representations can be combined after the particle pass has been completed, so each map is kept local to the PE and only at the end is the data combined. The particle data for the photon map is very easy to combine, since photons are just stored and later combined to form a kd-tree for each photon map or used to create an illumination map. Thus at the end of the particle tracing pass, the particle data for each photon map can be merged before the kd-tree is created. The illumination map is even easier to combine, since depending on how it is used, either a count of the number of particles hitting a patch on its surface or the energy deposited on a patch is stored. Thus these structures can be combined by adding the values stored in the patches together. Then both schemes can be normalised, as the scale factor is just a single number which can easily be communicated to all the other nodes. This approach suffers from the same limitations as the path tracing methods in that as the path length gets longer, more of the scene is traversed, thus effectively reducing its efficiency especially with large scenes.

5.6.5 Problems and Solutions

The biggest problem with the stratification method is that it has a hard time dealing with very long path lengths. However, since long paths do not contribute much to the final result (due to the light absorbed or lost via other means on each bounce), this is not a huge problem and there are a number of ways of getting around it. These involve using the illumination maps created by the

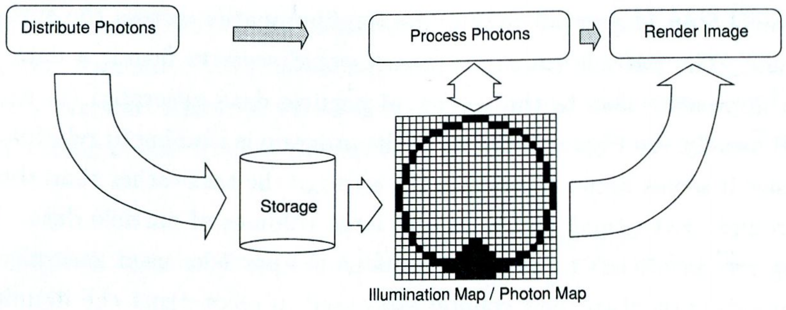


Figure 5.47: Algorithm Stages

first pass as light sources in the second, thus extending the length of the ray paths. This approach is similar in some ways to radiosity methods, which fire light from patches, treating all surfaces as light sources. In this manner the path length can be extended to arbitrary lengths without causing excessive paging. This has not been implemented as yet.

5.7 Markov Chain Monte Carlo Simulation

The MCMC methods are another group of methods which are widely used in computational physics. They are generally used to generate a series of samples from a complex function for which it is hard to determine a PDF. Unlike the previous two approaches it is not an adaptive scheme in that is does not learn about the function however it does respond to changes in the values returned by the function. The algorithm consists of three phases (see Figure 5.47) [78]: First a particle tracing phase shoots particles into the scene and representative data is gathered. Next follows a processing phase where this particle data is converted into a form that can be used during the rendering phase. Finally the rendering phase is used to display the final image. This structure is similar to framework presented in [131].

5.7.1 Particle Tracing Phase

The algorithm uses the Metropolis-Hastings sampling method described in a previous chapter with either the potential or rendering equation to build up a series of samples that express the illumination across a surface. Since the Metropolis method generates the samples according to the potential function $W(x, \omega)$, the distribution of the particles should converge to the distribution of real photons. Hence, by obtaining the density of these particles, the flux on a surface can be estimated.

A simple algorithm for the Metropolis method was presented in section 3.1.5

the method used a mutation function to generate new samples. Unfortunately, using this Metropolis sampling method directly with an integral over path space proves to be very slow. This is because for each sample generated the entire function must be evaluated, thus n ray queries are needed which slow down the calculations. Instead, the integral must be broken down into smaller subparts which can be executed relatively quickly. The Metropolis method should execute one ray query per mutation if its efficiency is to be comparable with other methods. Ray paths are parameterised by a set of variables:

- direction (θ, ϕ) for each ray i in the path
- wavelength λ
- film position (u, v)
- path length l.

Mutating these directly one variable at a time is inefficient, so a path is mutated by changing only its tail or final ray instead. This can be done by removing the tail, adding a new tail or changing the current one. To do this the following parameters are displaced:

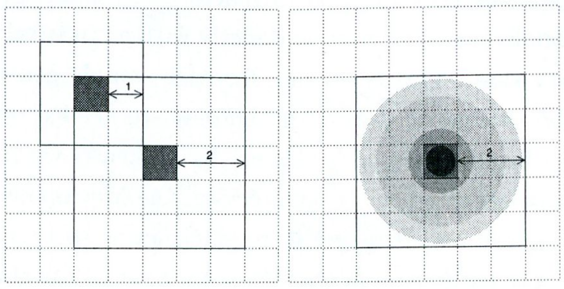
1. primary ray {path length l,wavelength λ, direction (θ, ϕ), film (u, v)}
2. other rays {path length l, direction (θ, ϕ) }

1 is only used if the path length is one and 2, is used in all other cases. l allows us to alter the path length and hence increases, decreases or keeps the path length the same. This is always performed first following which the subsequent values are used to perturb the tail produced. This way of mutating the path obeys the detailed balance condition of equation 3.31 because the probability of going from A to B is the same as going from B to A. Each tail is mutated by a set portion of the extents of its parameter domain as follows

$$x_{j_{i+1}} = (2\Delta x \zeta_j - \Delta x) + x_{j_i} \tag{5.26}$$

where x_{j_i} denotes element j of x produced on mutation i, ζ_j is a random variable generated for element j and Δx the maximum displacement amount for element j. In order to keep the number of ray queries at a minimum, the path length can only be changed by $\{-1, 0, 1\}$ and does not follow the continuous process of equation 5.26. If any of the mutated values step outside the bounds of their parameter range, the entire sample must be rejected and the previous value used.

The algorithm can use multiple Markov Chains to avoid getting stuck in any region, thus increasing the effective convergence to the real distribution with



(a) Variable Kernel (b) Fixed kernel

Figure 5.48: Types of Support

difficult scenes. Unfortunately, this has the side effect of wasting more samples due to the start-up bias. To reduce the number of wasted samples the step size must be increased giving noisier results. Ideally one wants the smallest step size possible to reduce the noise, but this increases the correlation between successive samples, thus in turn increasing the number of samples before convergence. Determining the number of samples to reject for start-up bias is difficult at best. A good rule of thumb is as follows: If the largest scale of the parameter space is L, and the proposal distribution $A(a \to b)$ has a maximum step size of ϵ, then at least $T \simeq (L/\epsilon)^2$ iterations are needed to obtain an independent sample. This agrees with much of the MCMC (Markov Chain Monte Carlo) literature.

5.7.2 Processing Phase

During this phase the stored photons are converted from clusters of photons into a data structure which can be easily and quickly referenced by the rendering method. Two different approaches to representing the illumination across the surfaces of the scene have been implemented: the photon map and the illumination map. Due to the fact that the storage costs of a photon map tend to be relatively high, illumination maps are used for any surfaces where it is possible to do so. However, the photon maps NN(Nearest Neighbour) technique is ideally suited to the Metropolis density estimation method. This is because less photons will be distributed in poorly lit areas and high numbers of photons will be found in bright areas and, as mentioned earlier, each photon carries the same energy. Thus effects such as caustics will come out more accurately as they have will have a large number of particles to represent them, as a side effect areas which are not very bright will have fewer particles and therefore less definition. Uniform grid illumination maps do not pick up artefacts such as

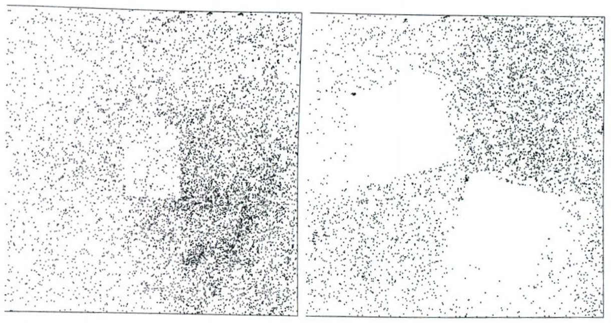

(a) Roof (b) Floor

Figure 5.49: Particle Distribution for MCMC Method for Cornell Box

caustics well unless they are of a very high resolution, and high resolution maps leave areas of low photon densities looking very noisy (see Figures 5.49 and 5.50). To compensate for this, a NN-like method is used as a post-process on the illumination map in order to reduce this effect. This makes the low density areas use bigger support areas, effectively lowering the resolution of the map in these areas. However, a suitable resolution still must be chosen, otherwise too much memory is wasted or not enough detail is obtained. To implement these post-processes we use two different schemes (see Figure 5.48):

- Fixed Kernel
- Variable Kernel

These methods have been used before by Collins [20] and Walter [132]. The Fixed Kernel method uses one of the kernels described earlier to "paint" a sample onto the illumination map using a fixed bandwidth. The variable kernel scheme works by continually increasing the size of the support area according to the local density (these methods are described in a previous chapter and also in [121]). In order to speed up the variable kernel method, a fixed kernel or histogram method is used in a first pass to render the local density into a buffer of the same or different resolution as the illumination map. These values are then used as the local density for the variable kernel method. Finally, to convert the given density to a flux, the density is multiplied by the total power of all the light sources.

5.7.3 Rendering Phase

To render the final image a pure path tracing method combined with quasi-random sampling or one of the adaptive schemes is used to remove any noise from

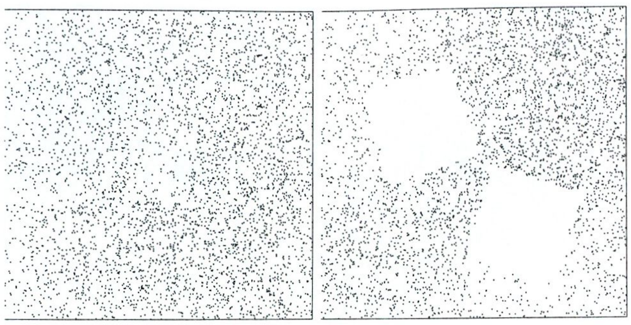

(a) Roof (b) Floor

Figure 5.50: Particle Distribution for Random Sampling Method for Cornell Box

the image. This phase works by performing path tracing until an illumination map or photon map is hit, at which stage the flux is read from the map and the resulting irradiance returned. If greater accuracy or definition is needed the illumination maps can be used only after the n^{th} ray bounce.

5.7.4 KD-Trees and MCMC methods

Using the previously described method with the photon map presents a number of problems. Normally when using a photon map with random sampling, the particles are very rarely the same because they are distributed randomly. Therefore, it is relatively safe to assume that no particles occupy the same position. If they do, their power can simply be combined to produce one particle. The MCMC methods do not distribute particles randomly, they reuse samples in order to distribute them according to a specified function. This means that there will be many particles which are identical. To handle this situation an extra field has to be added to the kd-tree to record duplicates. This is similar to the power field used with the random sampling method except it is an integer. Unfortunately, this increases the size of the event structure, but since more than 20 rejections is unusual, 1 byte is sufficient to represent this quantity. As there can be many duplicate particles, it means there will not be as many particle instances recorded as with the standard method, where nearly all particles are different. To keep the size of the photon map to a minimum, a number of different representations for photons are allowed, depending on the properties of the surface assigned to the map. If a surface parameterisation is available, a 2 dimensional coordinate scheme can be used to identify particle positions. For diffuse surfaces, the incoming direction of the particles can be removed.

```
struct{
          float x,y,z;              // surface position (12 bytes)
          char tag;                 // discriminator (1 byte)
          char count;               // number of particles (1 byte)
          char theta,phi;           // incoming direction (2 bytes)
          char wavelength;          // wavelength (1 byte)
        } glossy_photon_big;
struct{
          float u,v;                // surface position (8 bytes)
          char tag;                 // discriminator (1 byte)
          char count;               // number of particles (1 byte)
          char theta,phi;           // incoming direction (2 bytes)
          char wavelength;          // wavelength (1 byte)
        } glossy_photon_small;
struct{
          float x,y,z;              // surface position (12 bytes)
          char tag;                 // discriminator (1 byte)
          char count;               // number of particles (1 byte)
          char wavelength;          // wavelength (1 byte)
        } diffuse_photon_big;
struct{
          float u,v;                // surface position (8 bytes)
          char tag;                 // discriminator (1 byte)
          char count;               // number of particles (1 byte)
          char wavelength;          // wavelength (1 byte)
        } diffuse_photon_small;
```

Figure 5.51: Examples of photons structures

These small changes can result in big differences in the sizes of the photon maps produced and therefore the number of particles that can be stored. Figure 5.51 shows a selection of some of the different particle representations used. Note the fact that glossy particles or illumination maps don't store the exact incoming direction but a discrete version, because there is rarely a need for such accuracy except in the case of specular surfaces. These are better solved using pure path tracing and therefore not good candidates for use with an illumination map.

When using density estimation techniques near boundaries such as the edge of a polygon, more samples tend to be drawn from one side of the query point because more samples are available from there, due to the query radius crossing the boundary of the polygon. This leads to an increase in the size of the support area and thus lowers the density, causing inaccurate estimates to be made where the region of influence extends beyond the edge of the sample domain (see Figure 5.52). This is known as the boundary bias and for most statistical purposes is not a huge problem as it is generally the centre of the domain that is of interest. For global illumination, it causes problems because the whole

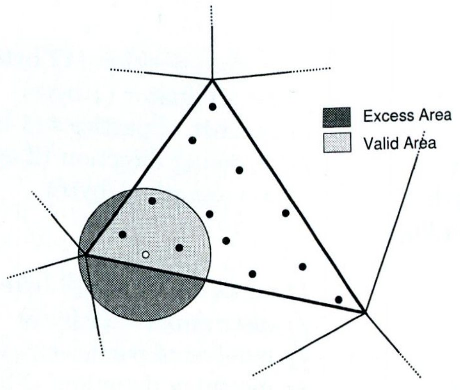


Figure 5.52: Boundary Bias

domain is important. The effect can be reduced by adding more samples, but it can never be eliminated by doing this. In images which use the photon map to represent the illumination across a surface, it shows up as a dark band near the edges of surfaces. Walter [132] has presented a solution to this edge effect which uses a density estimation technique based on linear regression.

5.7.5 Film Plane Metropolis Density Estimation

A novel use of the MCMC methods is to produce a density map for the film, instead of on the surfaces of the scene. A resolution independent representation of the "image" can be generated. Although the images may appear blurry relative to more classical integration based approaches, as more samples are added, this effect disappears and provides an implicit form of anti-aliasing. The user is also given a higher degree of control over the number of samples used to create an image. Now, instead of the usual $width \times height \times n$ number of samples needed, the exact number used can be specified. Avoiding this dimensional explosion was one of the main reasons for using Monte Carlo approaches in the first place. This scheme is similar to that of Veach [130], but instead of using integration methods, density estimation is used. A similar scheme that used integration instead of density estimation was presented by Dutre [31]. This suffered from dimensional explosion and produced poor results. Using density estimation on the film is not a new idea either, as Suykens [35] proposed a 2 pass filtering scheme using the Epanechnikov kernel. Perhaps a more compelling advantage to using this technique is that now brighter areas tend to get more samples than areas which are relatively dark. Thus if this is combined with tone operators and other similar schemes, samples should be directed to areas which are important to the viewer. Additionally, the sheer number of samples that can be used to solve one problem may create a better solution than integration based

```
struct{
        float u,v;          // surface position (8 bytes)
        char tag;           // discriminator (1 byte)
        char count;         // number of particles (1 byte)
    } image_sample;
```

Figure 5.53: Film plane sample structure

methods. Previously, relatively small numbers of samples were used to solve a number of sub-problems (integration on the pixel), compared to what other sciences such as physics use with Monte Carlo methods. This effectively means that the law of large numbers is not applicable to the schemes, which makes the error bounds greater because they are now bound by the Chebychev inequality and the methods are therefore less effective. Lastly, as the photons are being stored on a single surface, only a 2D representation need be used to record the photon's location, and the wavelength need not be recorded as the process can work in CIE_{XYZ} space using a photon map for each dimension. This provides the ability to store more photons and also reduces the size of the search space from 4 dimensions to just 2. Thus the storage costs are just 10 bytes per photon instance, which is smaller than for the other schemes previously described.

5.7.6 Sample Distribution Strategies

There are three possible strategies for developing an algorithm to distribute samples on the imaging plane. The first is to use the potential equation to parameterise particle paths originating from the light source and terminating on the film. The second approach involves using the rendering equation to start a ray path on the film and trying to join it to a light source via direct lighting. Thirdly, by combining these approaches using bi-directional path tracing (as in Veach's Metropolis light tracing), the benefits of the two previous approaches may be combined. The direct sampling approach, whereby a ray is fired at either the light source or the imaging plane, can in some situations be quite ineffective. This is especially true for pinhole cameras when used with light ray tracing, where the path to the film plane is very restricted. Each different approach has its merits which are similar to their integration based counterparts. This time however, enough samples can be used so that the law of large numbers is applicable. The process is also restartable so we can keep re-using old runs, thereby further improving the image. Since the particles can be stored different resolution images can be generated independent of the imaging process. However, as the number of samples increases, so do the storage costs, but particles can be saved to disk and then recalled when needed to create the final image. If the final image size is fixed, the samples can be added on the fly to the image thus requiring no intermediate storage. This results in a loss of flexibility, as the intermediate representation is lost, so no sophisticated density estimation methods can be used to render the image, such as the variable kernel method.

5.7.7 Mutation Strategies

The possible effective mutation strategies for this use of the Metropolis method are far larger than that of the previous methods. The only real restriction that limits the mutation strategies is that it is more beneficial if the ray intersecting the imaging plane is moved frequently. This stops the algorithm focusing too much on a single pixel or area. The simplest possible strategy is to perturb all elements of the parameter space by a random amount, thus guaranteeing that the position of the sample on the image plane is only infrequently the same value. As a side effect, it causes very large changes to the ray path for anything but very small changes to the parameters with eye path based schemes. Using light path tracing therefore seems a good idea, as only the ray incident on the film need be mutated each time. This allows a smaller number of ray queries to be made, similar to the view independent approach.

5.7.8 Handling the Boundary Bias

As mentioned previously, the boundary bias causes visible artifacts in pictures which use density estimation methods. For the film density estimation method there is a simple and effective solution: The film plane is made larger than needed, and the excess is cut away thereby removing the incorrectly estimated areas. Thus the estimates are generated correctly and do not show any boundary bias artifacts. There are two ways of doing this, the first being to just cut away the dark band around the image once it has been produced. This will waste some computation time but is tiny compared to the time taken to generate the image. The second method is to generate the image in an area smaller than the image plane such, that any density queries will always fit on the image plane.

5.7.9 Parallel Computation with MCMC

A parallel implementation of the Metropolis method is possible by running multiple chains simultaneously on separate PEs. Coherency is improved by allowing only small changes or perturbations to the parameter. This stops the rays from randomly accessing the scene and makes them gradually travel through the scene in small steps thus increasing coherency. The smaller the change the greater the coherency, but more evaluations are needed before samples are drawn from the actual PDF thus necessitating a longer startup period.

5.7.10 Compressing stored data

The random nature of the data generated by Monte Carlo sampling appears at first to be unsuitable for use with data compression techniques. However, data produced by the Metropolis technique is correlated and therefore susceptible to compression. The smaller the allowed perturbations, the more effective the compression will be as the numbers will only vary in their least significant digits, thus producing repeating groups of bytes (see Figure 5.54). Additionally, the count element stored with each data element contains many repeated strings, and is compressible with a simple run length encoding [138, 27] scheme. Figure 5.55 and 5.56 show the distribution of counts for a sitting room scene and the Cornell box. This can be further improved by rearranging the data. Many compression algorithms use rearrangement to improve compression but none did so successfully for our case. Using rearrangement and the already present correlation, the data can be compressed using the *deflate* [28] algorithm which applies Huffman encoding [100, 29] and LW77. The rearrangement of the data makes it easier for the algorithm to identify repeated items within the data stream. To do this, the data is saved as strips of x's, y's etc rather than as individual particles. Thus repeated entities are close together (see Figure 5.57). For instance, the wavelengths for n particles are saved together, thus repeated strings of the same wavelength occur for particles in the same ray path. The compression algorithm can easily identify these, and affords some compression even for completely random sampling. This is very minimal because the wavelength is encoded into a byte and represents only a small portion of the particle data. Thus the storage costs are reduced as well as the time spent on I/O.

Rather than rewrite this code, the zlib [81] compression library used by the gzip program and the bz2lib used with the bzip2 program have been used. The bz2lib library tends to give a better compression ratio. The algorithm uses buffers to store the data until a user specified amount is reached, at which time the data is compressed and written to disk. Usually this is set at 2MB and when full, the data is saved as strips of each element. The is especially effective for the particle count as there tends to be a large sequence of ones which compress very well, even using a simple run length encoding scheme. So far, compression has not been used with the kd-tree as it makes it very slow. Perhaps a caching scheme would help improve the performance, but for large numbers of data items the kd-tree becomes prohibitively slow anyway.

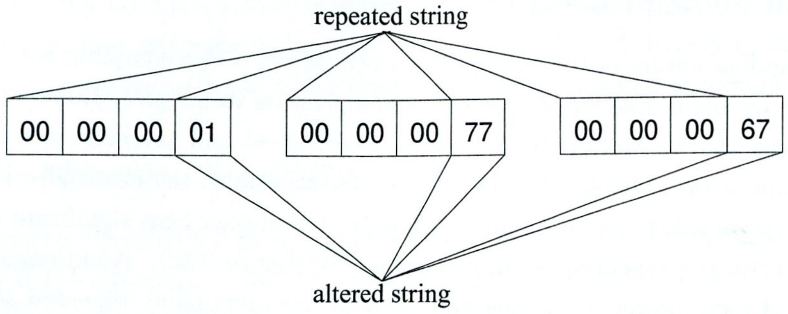


Figure 5.54: Correlation causes repeated strings which can be compressed

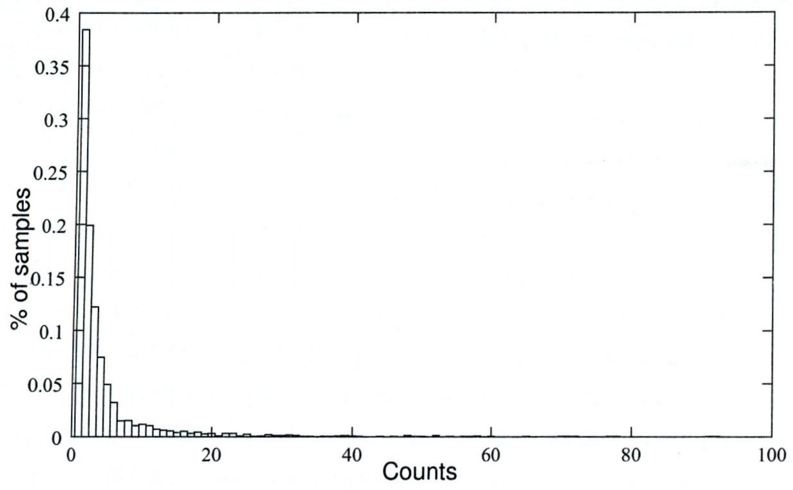


Figure 5.55: Distribution of counts for the sitting room scene

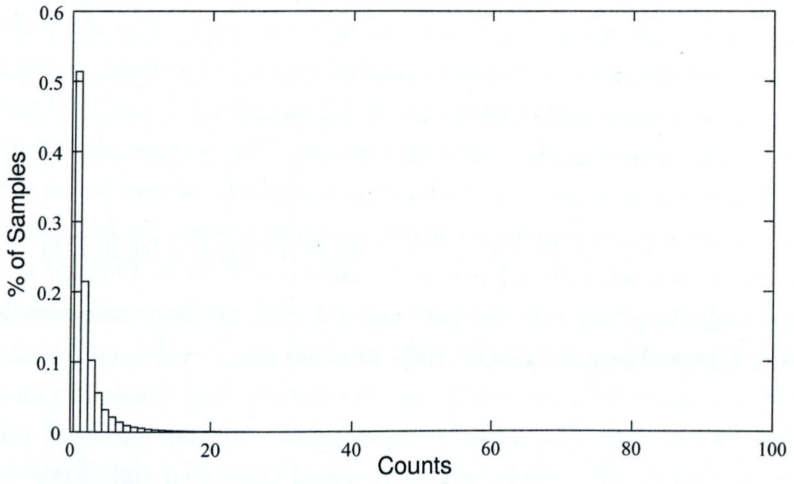


Figure 5.56: Distribution of counts for the Cornell box scene

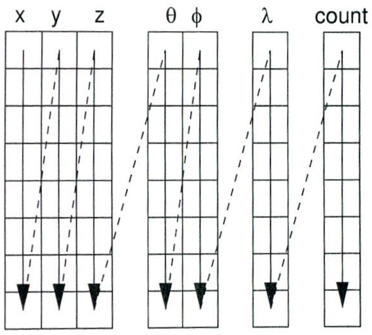


Figure 5.57: Data arranged as rows in memory is saved as strips

Chapter 6

Results

6.1 Rendering Framework

EFFIGI has been successfully used to implement many of the latest techniques in image synthesis including Monte Carlo techniques such as the Metropolis method and bidirectional path tracing. It is extremely flexible and provides a means of easy augmentation using both source code and shared libraries. The use of interfaces in the framework allows easy extension of many older methods without having to completely re-implement them. As all the modules can be provided as a shared library, only the modules actually referenced are loaded thus reducing the memory footprint of the system.

Implementers are also allowed to experiment with various ideas (for example sampling schemes) without having to alter any code. If the code is written in a generic way it can be used with many of the other components, thus extending the entire system. This encourages the user to break up any objects they create into many reusable parts so that they can be used with the other elements of the system. This has resulted in greater flexibility, more code reuse, and a powerful rendering environment.

Comparisons

Although comparing frameworks is difficult, the following attempts to highlight some of the differences between our framework and two others introduced in Section 5.2.1 which share similar objectives.

RenderPark is an object oriented rendering framework which is freely available to the public. Both it and EFFIGI divide the set of rendering techniques into view dependent or view independent, and this dictates how they are implemented. The strength of RenderPark is its extensive suite of Radiosity algorithms. However, it lacks the mathematical basis of EFFIGI, meaning that integration and other mathematical methods cannot be changed easily. In addition, scene acceleration has been implemented using utility functions that do not support any common form of interface, and as yet this system does not support texture maps, only allowing a single colour and BRDF to be assigned to a surface.

Vision is another object-oriented rendering framework which, like EFFIGI, derives its geometry subsystem from the Ray Tracing Kernel [67]. Vision uses a physically based object-oriented abstraction of the rendering process, and can render both physically and non-physically correct systems. However, it specifies fixed mathematical models for some interfaces that cannot be changed by the programmer, for instance the VolumeLighting subsystem. In addition, the sampling interfaces have been fragmented for use with different techniques.

Table 6.1 presents a summary of the features in the frameworks: EFFIGI, Vision and RenderPark.

features	EFFIGI	Vision	Render Park
Physically based	yes	yes	yes
Radiosity	Progressive Refinement	many	many
Object Oriented	yes	yes	mostly
Plug-ins	yes	yes	no
Mathematical basis	yes	no	no
Object Model	COM	CORBA	no
Parallel Computation	yes	no	no
Particle Tracing and density estimation	many	no	some
View Dependent Methods	many	many	many

Table 6.1: Framework Features

6.2 Analysis of Methods of Solution

In this section the various algorithms described in the previous chapter will be analysed to see if they improve the efficiency and convergence of the global illumination process. A number of test scenes are used to create a variety of scenarios in which to test the algorithms.

6.2.1 Error Analysis

To show the rate of convergence and the approximate error of the techniques described previously, the root mean square (RMS) error is used. The RMS error is calculated as follows:

$$\sigma = \sqrt{\frac{\sum_{p} \left[L_{ref_p} - L_p\right]^2}{N}}, \tag{6.1}$$

where N is the number of pixels or regions, L_p is the value of a region p in the image being tested, and L_{ref_p} is the corresponding value of that region in the ideal image. By graphing this quantity with respect to the number of samples, the rate of convergence can be shown and by graphing with respect to the time taken, the efficiency of the scheme can be illustrated. These are the main metrics that are used to show the superiority of one algorithm over another. To generate the image maps necessary to be able to use this algorithm the illumination maps are rendered into one large array of values where each individual map is arranged in a linear fashion one after the other. For standard view dependent path tracing schemes the image buffer was used. In the case of the density estimation viewing methods the density map was just rendered to a buffer using a histogram.

The perceptual issues involved in accessing images or illumination maps have not been taken into account except to the extent that one image may appear smoother or less noisy than another. Analysing the perceptual issues involved are beyond the scope of this document.

6.2.2 Test Images

In order to assess the relative merits of each individual scheme and their combinations, a series of test images has been produced. These vary in complexity and try to test the algorithms in contrasting situations. Also, some standard scenes with which many people are familiar with have been reconstructed so as to provide a reasonable reference. The scenes used are as follows:

- Cornell Box: The Cornell box [46] is a very simple scene consisting of two different coloured and sized boxes standing in a cube containing a square light on the ceiling. This is a classic radiosity test scene used to demonstrate colour bleeding effects. The entire scene usually consists of diffuse surfaces but sometimes the taller box is made specular. This produces a caustic on the ceiling. The two versions of the Cornell box are shown in Figure 6.1.

- Ring: The ring scene consists of a silver ring sitting on the floor in a cuboid shaped room which produces an epicycloid shaped caustic on the floor due to a light on one of the walls (see Figure 6.2). This is another classic scene used to test how well a rendering system handles caustics.

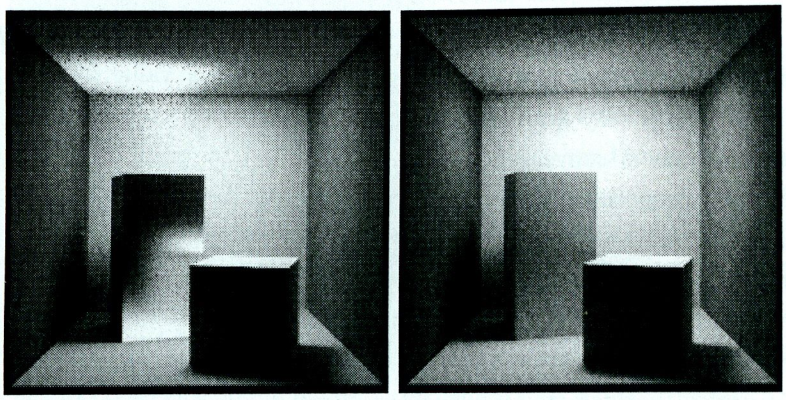

(a) Specular (b) Diffuse

Figure 6.1: Cornell Boxes

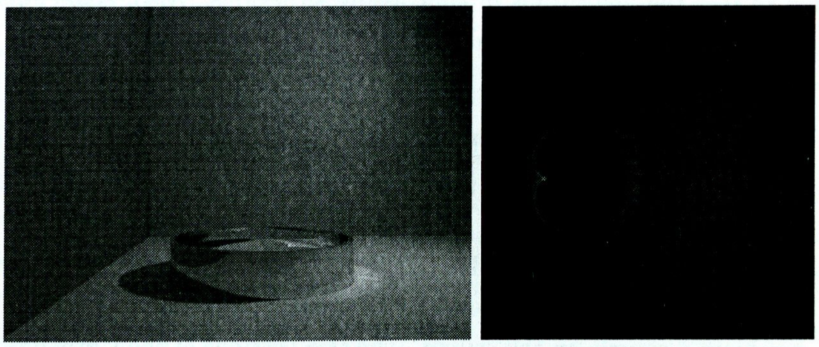


Figure 6.2: Right: Ring consisting of 1028 polygons. Left: Illumination Map from floor.

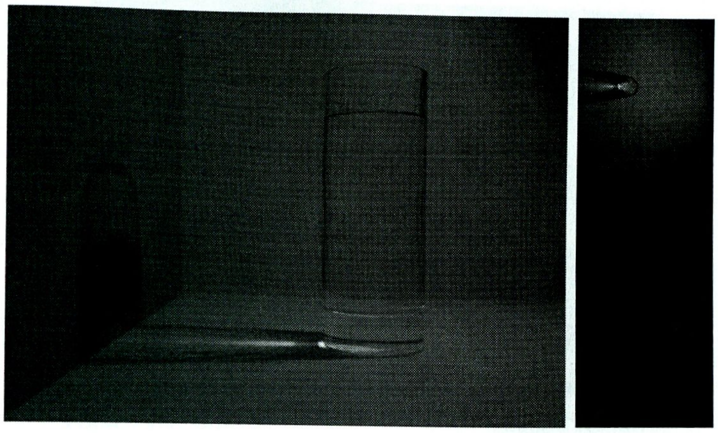


Figure 6.3: Right: glass of water consisting of 14,613 polygons Image. Left: Illumination Map from the floor.

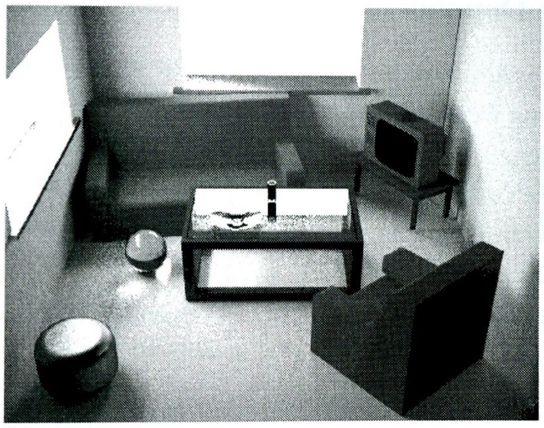


Figure 6.4: Sitting Room

- Glass of Water: As another test of a renderer's ability to produce caustics, a large glass of water standing on the floor of yet another cube shaped room was rendered. A light on the far wall casts a caustic on the floor and wall on the opposite side of the box (see Figure 6.3).

- Sitting Room: This is a more complex scene consisting of a sofa, an armchair and a table with a glass top, facing a television set (see Figure 6.4). There are two windows in the scene which are used to illuminate the room. The detail in the room is very high, as even the aerial and buttons on the television are present, along with plug sockets and power leads.

- Office scene: A complex diffuse environment with many indirect lighting effects is shown in Figure 6.5. This scene was provided by Leo Talbot.

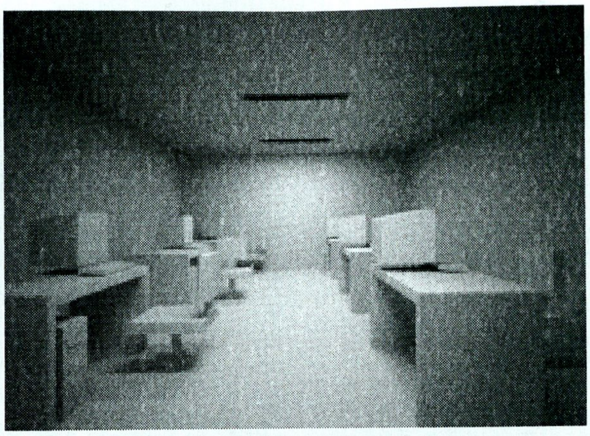


Figure 6.5: Office

6.2.3 View Dependent Methods

This section compares the three view dependent methods with naïve Monte Carlo path tracing for a number of scenes. The graphs of convergence results are shown in Figures 6.6, 6.7 and 6.8. These clearly show the convergence of each scheme. The reference scenes were all generated using about 16384 samples per pixel (the average number of samples to render a picture in 12 hours i.e. a night), with a resolution of 100×100 for the Cornell box scenes and 128×91 (this gives the image the aspect ratio of normal 35mm camera photograph) for the sitting room scene. All results were generated using a PII 350Mhz with 128MB RAM running FreeBSD.

The first graph shown in Figure 6.6 shows the convergence of the three new methods presented against standard random sampling. The poor initial results for the VEGAS method are because a number of samples, 800 in this case, are needed before it can properly adapt. Since the random sampling already has a relatively good local sampling scheme, due to cosine sampling it performs better initially. The Stratified and MCMC methods do very well. However the stratified method is constrained by the number of samples it requires, due to combinatorial explosion, but it adapts better initially due to the better spread of samples through the problem domain.

In order to test a more difficult scene, a BRDF was used without its sampling scheme. This was done by using the Phong BRDF on the tall box in the specular Cornell box scene. This tests the algorithm's ability to adapt in comparison with a standard path tracer. The graph in figure 6.7 shows that the adaptive algorithms and the MCMC algorithms perform far better than random sampling. The MCMC algorithm in particular does very well, because it can place the samples wherever it wishes, thus focusing more effort on problem

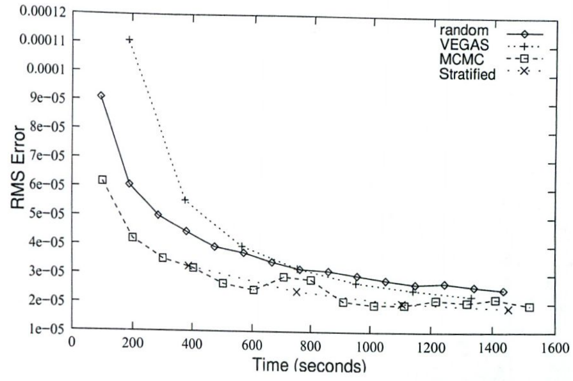


Figure 6.6: Diffuse Cornell Box Convergence Results

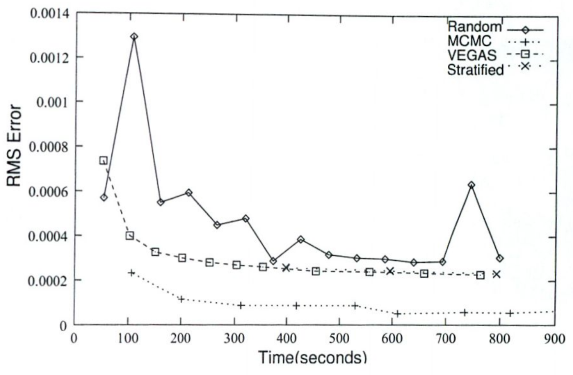


Figure 6.7: Specular Cornell Box Convergence Results

areas.

The sitting room was used to test the algorithms in a more complex environment. The graph in Figure 6.8 shows that the three new algorithms performed equally well. Once again the stratified algorithm is restricted by the number of samples it must use, hence the limited number of points. It would appear that the more complex the scene, the better these algorithms do, and this seems to hold for other scenes rendered. The VEGAS and Metropolis techniques are very effective for reducing the apparent noise in scenes, especially where no useful importance sampling schemes are available. Figure 6.9 shows the pictures generated by the these two methods and one using naïve Monte Carlo path tracing. A table comparing all the view dependent methods is shown in table 6.2 and 6.3. Notice that although the VEGAS and Metropolis methods have the same error characteristics for the sitting room the pictures are very differet. This is because VEGAS algorithm puts an equal amount of work into each pixel but the MCMC one does not. The MCMC algorithm, because of the way it samples the problem domain, is strongly attracted to lights. This can be a problem for some scenes, because samples are drawn away from other areas

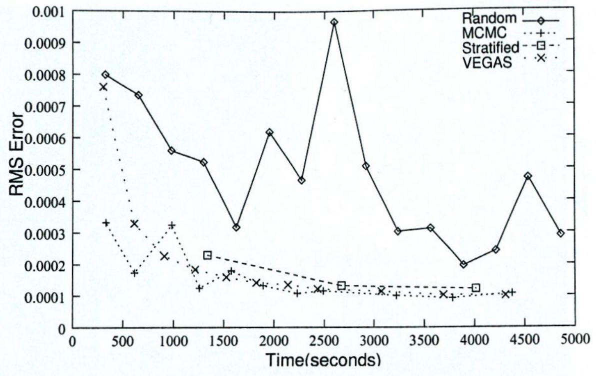
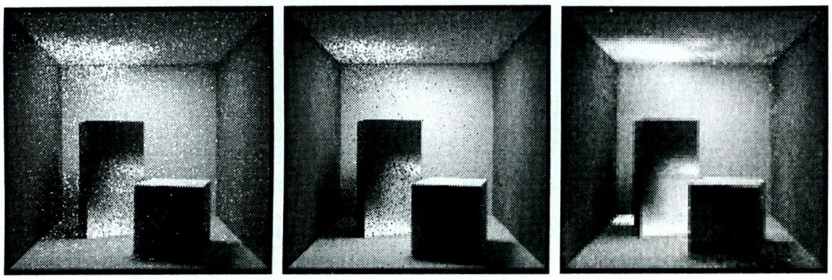


Figure 6.8: Sitting Room Convergence Results



(a) Naïve (b) VEGAS (c) Metropolis

Figure 6.9: Cornell Boxes generated using different methods taking about 23 minutes each

to sample the light source, which can be wasteful in some situations. There is a number of solutions to this problem: One is to split the image into two regions, those with light sources and those without. This can produce a better distribution of samples where the light sources cause a problem; Another method is to ignore lights when hit by the primary or eye ray and use a separate pass to "paint" them in later.

6.2.4 View Independent Methods

This section compares the ability of the two particle tracing methods to create illumination maps. To do, this the two Cornell boxes, the ring and the office scene with a glossy floor are used. For the Cornell box scenes, an illumination map was applied to all surfaces with a resolution of 50. For the ring scene, illumination maps were used with a resolution of 50 × 50 but no map was applied to the ring because it is specular and the map was therefore not necessary, as pure path tracing is better suited to its solution. Finally, for the office, the

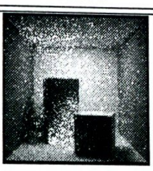
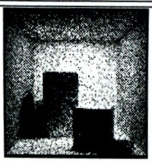
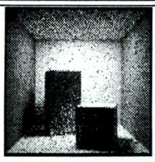
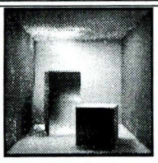
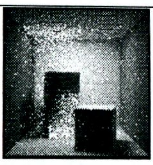
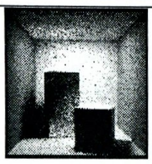
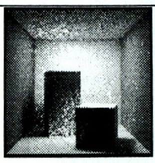
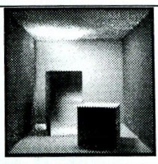
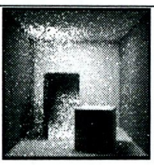
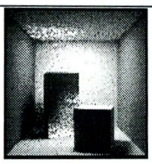
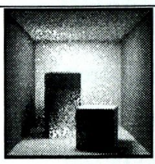
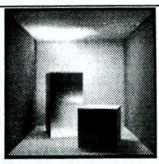
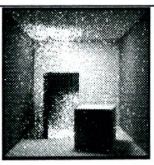
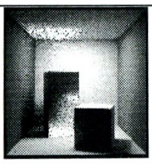
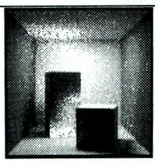
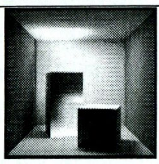
Samples	Naïve	VEGAS	Stratified	Metropolis
256				
512				
1024				
2048				

Table 6.2: Visual Comparison using differing methods for Specular Cornell Box

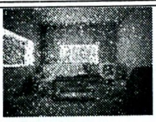
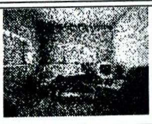
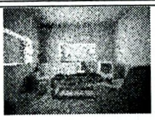
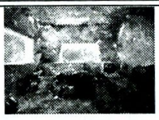
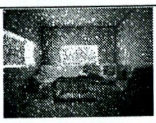
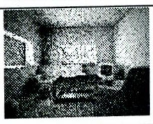
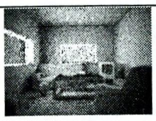
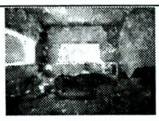
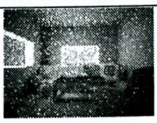
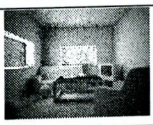
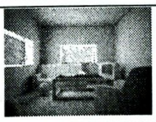
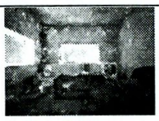
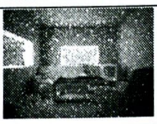
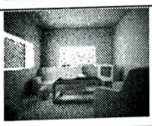
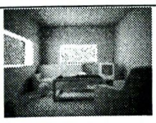
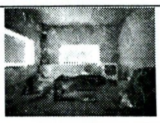
Samples	Naïve	VEGAS	Stratified	Metropolis
256				
512				
1024				
2048				

Table 6.3: Visual Comparison using differing methods for Sitting Room

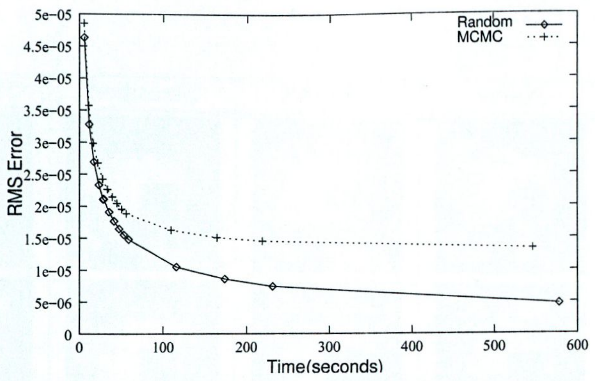


Figure 6.10: Particle Tracing Diffuse Box Convergence Results

minimum illumination texel size was specified as 0.005 and every surface was assigned an illumination map producing 9060 texels in total. All the reference illumination maps were created using 1024 million particles with standard particle tracing. The results were generated using a PII 350Mhz with 128MB RAM running FreeBSD.

The diffuse Cornell Box scene was used to test the MCMC algorithm in an environment that the standard particle tracing algorithm can handle very well, due to efficient sampling strategies. As expected, it performed better than the MCMC algorithm (see Figure 6.10). As a second test, the specular Cornell box was used, with the sampling strategy for the specular box removed. The standard particle tracer was still better (see Figure 6.11), but the difference in efficiency had narrowed. This may seem surprising at first, but since the interaction in the box is largely diffuse and the light is almost directly over the glossy surface, the cosine sampling used by default works quite well. Next, for the ring scene, the results for the MCMC were better initially but converged to the same value (see Figure 6.12). Thus there is no real advantage gained by using the MCMC algorithm with large numbers of particles for this scene. In order to create a better test, the office scene was altered so that the floor was now glossy, thus creating a non-ideal situation for the random sampler. As expected, the MCMC algorithm performed significantly better. Figure 6.14 shows these results. As a visual comparison of the results, the illumination maps are shown in Figure 6.13. These pictures used 10M particles to generate the illumination maps.

6.3 Parallel Analysis

In this section, the gains achieved by using stratification to divide up the scene between PEs are analysed. The graphs show the scalability of the algorithms.

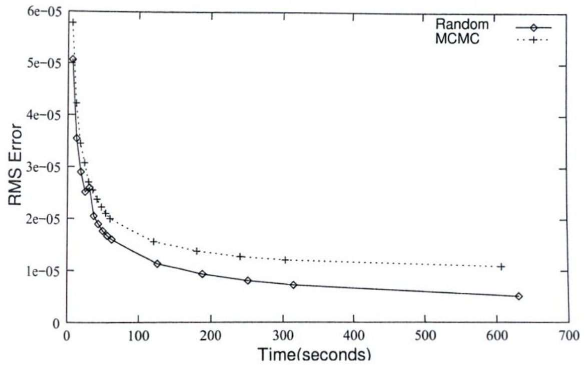


Figure 6.11: Particle Tracing Specular Box Convergence Results

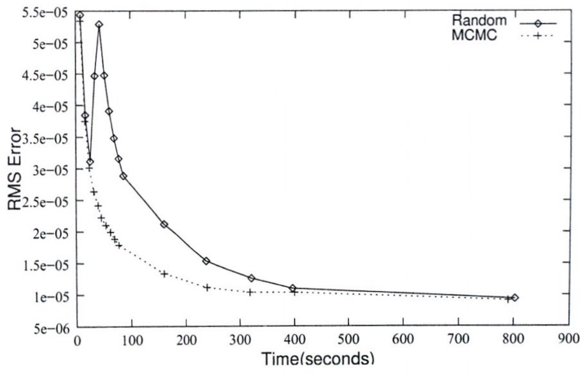
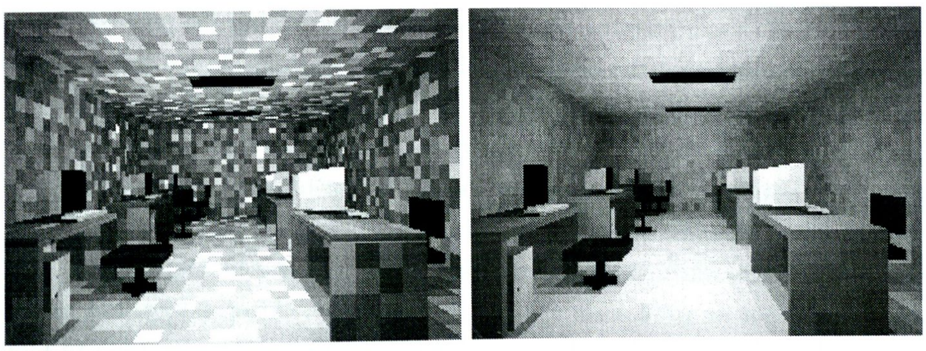


Figure 6.12: Particle Tracing Ring Convergence Results



(a) Random Sampling (b) MCMC Sampling

Figure 6.13: Comparison of Illumination Maps for Office scene with glossy floor using 10M particles

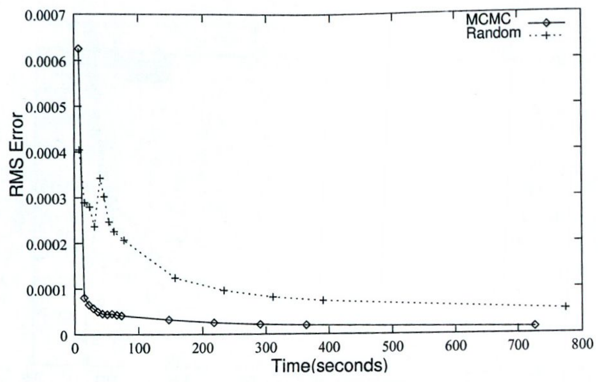


Figure 6.14: Office with glossy floor convergence results

The speed-up was calculated using the time to run the parallel algorithm on one processor. The scenes were rendered on a cluster of 16 PII 450Mhz with 256Mb RAM each running Redhat Linux.

The dependence of the algorithm on communication can be seen as the number of processors increases (see Figure 6.15 and 6.16). This is because as more processors are added, more data must be transfered. This data is either the image or the illumination maps. In the case of non-particle based representations, the amount of data increases by the number of processors, since the complete buffer must be transfered at the end. For these scenes the size of the image buffer was only 256 × 183 which is relatively small. Larger images would obviously incur a greater penalty. The amount of data transfered could be minimised by determining what surfaces were hit during the processing and then only transmitting the one which changed. The sitting room scene was an example of a view dependent method using parallel stratification. From the graph in Figure 6.15 it would appear that it is close to linear speed-up. The task distribution among the processors appears to be very even, as shown in Figure 6.4. The same stratification strategy was applied to a particle tracing solution for the office scene, and the speed-up is shown in Figure 6.16. The speed-up is not as good as in the previous example. After 4 PEs, the results seem to fall off a bit. This is because PE 5 appears to be in use (due to non-exclusive access to the cluster). This is verified by the task distribution where PE 5 has always completed less tasks (see Figure 6.4). The test was not re-run because it is very rare that no other tasks are using the cluster. It also serves to demonstrate the effectiveness of the scheduling algorithm in identifying load imbalances.

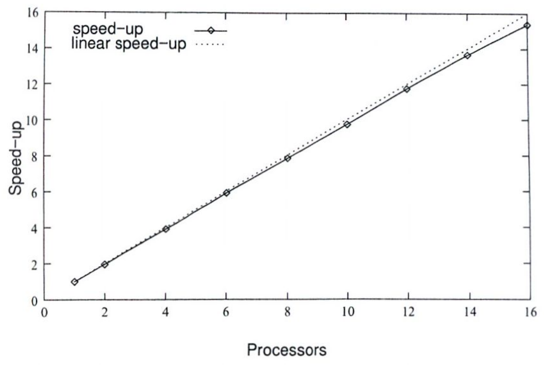


Figure 6.15: Sitting Room Speed-Up

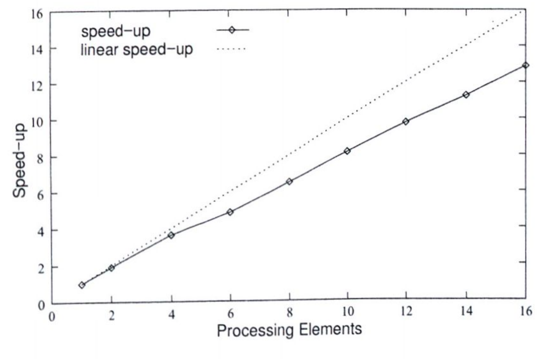


Figure 6.16: Office Speed-Up

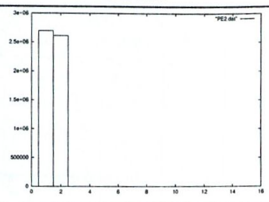
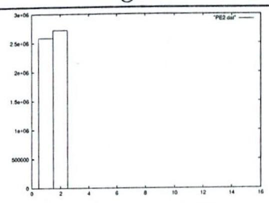
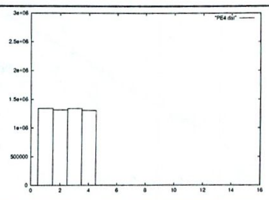
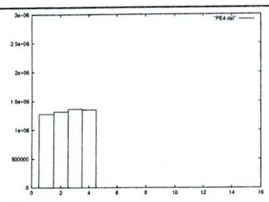
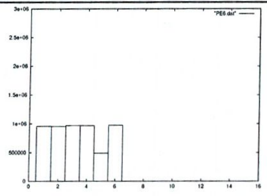
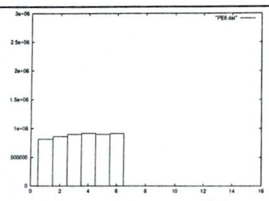
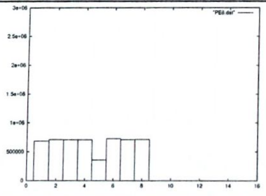
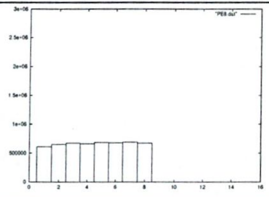
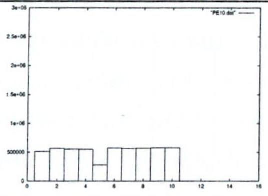
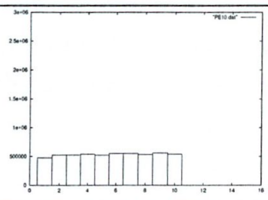
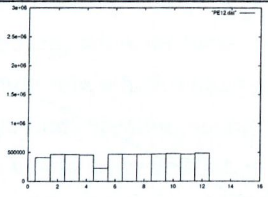
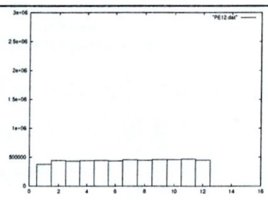
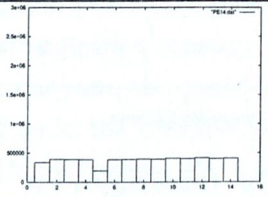
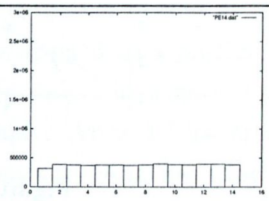
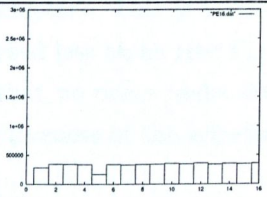
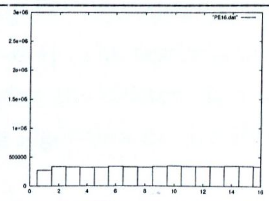
Processors	Office	Sitting Room
2		
4		
6		
8		
10		
12		
14		
16		

Table 6.4: Task Distribution for Sitting Room

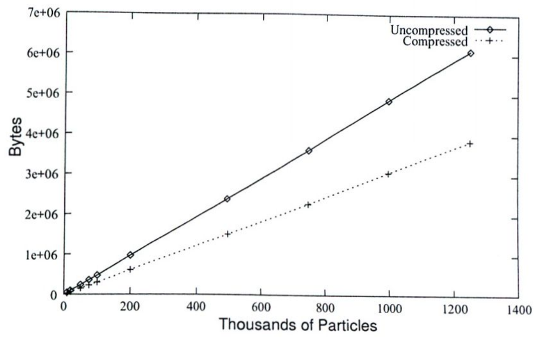


Figure 6.17: Compression Results For Cornell Box

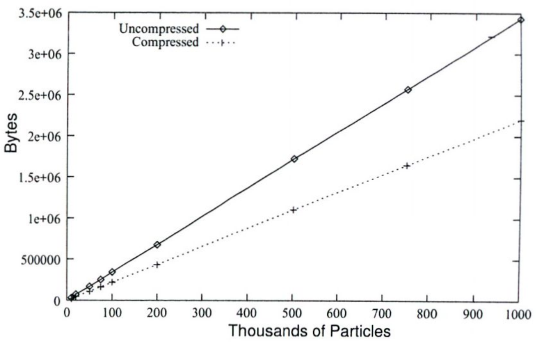


Figure 6.18: Compression Results For Sitting Room

6.4 Compression Analysis

We now analyse the benefits of compressing the data produced by the MCMC methods proposed, in terms of speed of execution and memory requirements. In order to demonstrate the benefits of compression, the rendering pipeline has been split into two stages: the particle gathering phase where the particle data is saved, and the rendering phase where the data is read from disk and converted to a suitable form such as an illumination map or a picture. The Cornell box and the sitting room scene have been used to obtain these results. The first set of results graph the amount of compression that is achieved, showing the relative sizes in bytes of compressed and uncompressed particle data. Figure 6.17 shows the results for the Cornell box and Figure 6.18 the results for the sitting room scene. These graphs clearly demonstrate the reduction of storage costs that is possible. In order to compress the data, some processing power is expended but the loss of processing power is negligible compared to the time required to load or save the data. Figures 6.19 and 6.20 show that the overheads required to save the data are virtually non-existent. Since writing to disk is a non-blocking operation, there will not be much time saved while writing data, unless very large chunks are written which overflow the buffers. Timing I/O operations is difficult

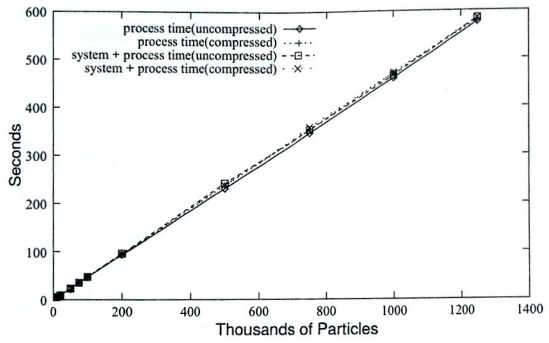


Figure 6.19: Saving Results For Cornell Box

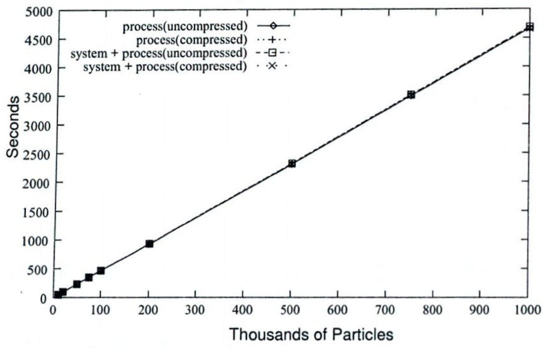


Figure 6.20: Saving Results For Sitting Room

as they are largely operating system(OS) controlled, thus to show the time spent writing and that spent processing, a graph of the process time and the real time taken is shown. This helps to highlight the processing time lost to compression versus that lost to I/O. As is evident, the buffering performs very well, causing hardly any delay when saving, just the time required to copy the data to a buffer. Despite the success of the buffering, the times for the compressed particle data are not very different. Obviously copying data into system space incurs some overheads. The final set of graphs shown in Figures 6.21 and 6.22 show the time taken to load and perform a histogram density estimation. The histogram was chosen because it was simple and fast and therefore would not interfere with the loading results we are interested in. Loading is a blocking I/O operation and thus buffering has little effect, so compression schemes should do well here. Using non-blocking reads would improve the results, but at least twice as much main memory would be required. Once again, the system and user time are shown to highlight the time spent processing and the time spent loading. The time spent loading has been reduced by a few seconds which increases with the number of particles.

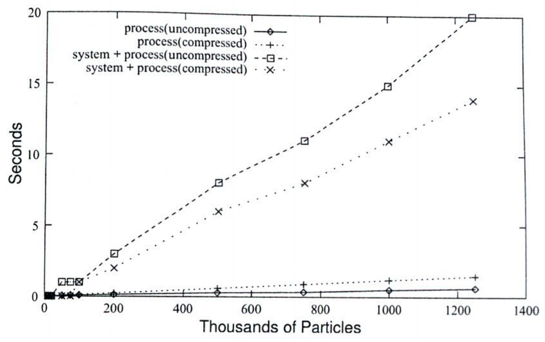


Figure 6.21: Loading Results For Cornell Box

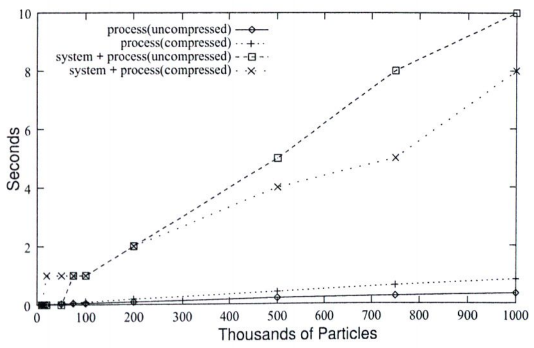


Figure 6.22: Loading Results For Sitting Room

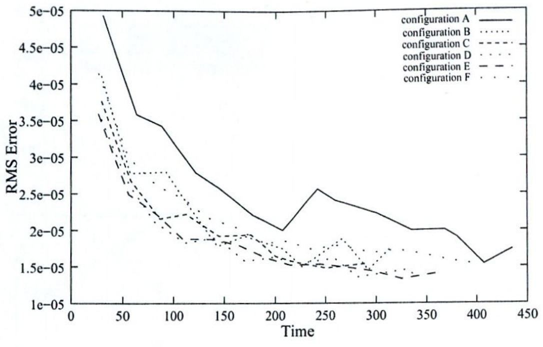


Figure 6.23: Result of Different MCMC configurations

6.5 Analysis of Results

The results show that the adaptive schemes for large numbers of samples are better than standard path tracing, even when importance sampling is applied. This is shown in Figures 6.6, 6.7 and 6.8. Other adpative schemes by Dutré [34], LaFortune [72] and Jensen [59] showed similar resuts but were constrined to adapt only to specific parts of the problem. This is clearly demonstrated by the first graph in Figure 6.6, where after a given number of samples the adaptive scheme becomes more effective than the naïve scheme. For difficult scenes especially when there is no good importance function available, the adaptive schemes are better in terms of both convergence and visual aspects (reduced noise in the image)(see Figures 6.7, 6.8 and 6.9). The Metropolis particle tracing scheme presented, although not better for scenes which have adequate importance sampling already provided, are better for difficult scenes for which such functions cannot be provided. In these cases, the visual noise in the illumination maps is reduced, and the convergence rate is significantly better than the naïve method (see Figures 6.14 and 6.13). When used on the film plane to create images of the scene, the Metropolis method is also very good, but it suffers from some drawbacks. The worst of these is its attraction to bright areas such as light sources. Although this reduces the overall error in the image, the visual impact is impaired. This is because tone operators [125, 76] usually map these regions to white. A second problem is that of configuring the Metropolis scheme, as even small changes to the mutation parameter can have big a impact on performance and also these parameters are different for each scene. Figure 6.23 shows a number of different configurations for the film Metropolis method on the glass scene. Light sources do not appear to be a problem in the particle tracing variant, because sampling a point on a light source has almost no cost, as no ray queries are needed. The results for the stratification approach for parallel computation show that it produces very good speed ups and can be applied to nearly all the techniques presented to some degree. The load balancing is good at evenly distributing the load between processors. One problem does remain, which is determining the best order to pick voxels such that coherency is maximised, which are currently chosen in index order.

Chapter 7

Conclusions and Future Work

The techniques presented in this thesis belong to a domain of computer graphics called realistic image synthesis. This involves the construction of a 2D image of a scene which could have been produced by a camera, given sufficient input data, in the form of scene geometry and surface properties. The principal contributions of this work are a framework which uses mathematical concepts, two different adaptive integration methods are used to solve the rendering and potential equations, a parallel rendering scheme is devised which uses stratified sampling, a Metropolis sampling method is created which uses density estimation rather than integration and finally a scheme for compressing the particles produced by the Metropolis sampling method.

7.1 Framework

First a framework is presented which enables the construction of algorithms and techniques suitable for global illumination. This breakup of components using mathematical concepts with which many users should be familiar, increases the reuseability of components because of this abstraction. The framework provides a modular and object oriented approach to constructing a rendering system. This was facilitated by using the underlying mathematical concepts as a basis for the design of interfaces within the framework. The component object model (COM) was used as a means of providing run time loading of components and type information which is language independent. To further improve the system, a simple scripting language was used to enable the configuration of many different rendering systems without the need for recompilation and also as a means of inputing scene and other data. To aid developers and researchers in the construction of new methods, various visualisation and analysis components were provided. These allow the user to easily see what the system is doing and therefore to improve error detection.

7.2 Integration Algorithms

Two adaptive Monte Carlo integration techniques were used to improve the efficiency of various tasks used in the rendering system. When used to integrate the rendering equation, the algorithms can adapt to complex scenes, directing the samples so that the error in the estimate is minimised. However, these schemes require the application of special considerations so that they can better predict the important regions. Complex ray queries which attempt to evaluate too many paths at once tend to confuse these schemes, so simplified path evaluation schemes are created which only evaluate a single ray path at a time. The dimensionality of the global illumination equations represent a problem for adaptive and quasi-random schemes. To reduce their dimensionality, intermediate data was produced using density estimation methods.

The first adaptive approach used is derived from the VEGAS algorithm. This scheme can use either a stratified or an importance based sampling strategy. Modifications were made to facilitate the needs of global illumination where complex integrals consisting of the evaluation of combinations of 1 to 4 dimensional sub-functions are required. The VEGAS algorithm tries to create a combination of sampling strategies which accounts for these functions. To do this a mixture of methods were used to divide the integration domain and highlight their significance.

The second method used stratified sampling to break up the domain of integration into a number of variable size segments which were assigned an equal number of samples. Unfortunately, due to the limited number of samples, only a small number of subdivisions were possible. Two approaches were used to divide the parameter domain: A blind stratification scheme which can be used to evenly divide the parameter space, or alternatively a number of samples can be used to estimate the variance and thus guide the subdivision and sample allocation.

These two schemes provide robust methods of solving global illumination problems, even when no suitable importance schemes can be used with the BRDFs of the scene. This allows users more freedom in the surface types they can render, as suitable sampling schemes are no longer a constraint, and enhances the systems ability to render complex and difficult scenes. The freedom in the range of surface models is because the adaptive schemes create their own sampling functions. They also adapt to the scene detecting areas of high and low contribution or variance, this makes them more efficient with complex scenes where the radiance is distributed less evenly.

7.3 Density Estimation methods

Two radically different methods are used to generate a distribution of particles on the surfaces of the scene. The first uses a blind stratification method to equally distribute the samples around the scene, thus each particle contributes a different amount, but the particles are evenly distributed through the environment. The other method uses the Metropolis method to deposit particles according to the potential equation. This results in the particles being of equal weight, which reduces the storage costs as their power is no longer recorded this reduces the photon storage size by about 4 bytes. However, they are no longer equally distributed but tend to congregate in areas where there is more light. A new mutation strategy for the directional parameterisation of the ray path is presented. This minimises the number of ray queries needed, thus making it practical to use. To create a path at most only one ray query is needed, rather than the n queries needed to create a entirely new path.

Classical integration based view independent methods allow only very few samples to be assigned to the estimation of individual pixels, which limits their effectiveness. An alternative method uses the Metropolis algorithm to distribute particles or samples on the image plane, allowing large numbers of samples to be used in one calculation. This method is used with many of the differing forms of path tracing. The image is produced using density estimation on the samples generated on the image plane. The algorithm can be run a number of times, thereby accumulating more particles and increasing the accuracy of the resulting image. Since it is resolution independent, the results generated from a small preview can be used. Due to the correlation of samples, compression can be used on the particle stream produced, allowing significantly more particles to be stored.

7.4 Parallel Methods

The coherency of ray path tracing was improved by using stratification to partition the ray path into segments, this limits there access to fixed smaller areas of the scene. These segments were then assigned to different processing elements. This method is used with either particle tracing or eye path tracing methods, with little or no modifications.

The Metropolis method is used in parallel methods by running a number of different chains on separate processors. By minimising the amount of mutation to a ray path, the amount of coherency can be increased because the change in ray path is smaller and hence take longer to change position, but at the cost of increasing the startup bias. This results in more samples being wasted before unbiased samples can be drawn from the chain.

7.5 Future Work

The rendering framework created an environment capable of implementing the many rendering algorithms and methods associated with image synthesis, and is not overly complicated. The user and not just the developer is able to use the environment and tailor it to their needs. To this end, the scripting language should be enhanced, so that more common high level languages such as LISP, Perl or Python can be used. This will allow both users and developers to improve the functionality of the program more easily. Adding the Component Object Request Broker Architecture [56] (CORBA) and the Distributed Component Object Model [110] (DCOM) to the framework would also prove very beneficial as many people do not like one or the other of these systems. These systems allow the use of components or objects on remote machines, with out the programmer or user having to be aware of this. Therfore CORBA and DCOM would allow remote management and control of the system so a user need no longer access the render farm directly. Additionally, the ability to use the framework from remote locations would be useful.

Every global illumination researcher working on rendering algorithms hopes to produce a single algorithm which can solve all rendering problems effortlessly. To date no such technique exists. Even the Metropolis method which was a holy grail for a time has its pitfalls. The main problem lies in the fact that only very small numbers of samples are used to solve these problems (even a thousand samples is a very small number). If the $n \times m$ subproblems of determining the colour of each pixel are replaced with one big problem, the law of large numbers will be applicable. With this in mind, adaptive algorithms, which barring a few papers have been largely ignored to date, will become far more important. Using the VEGAS importance sampling method with particle tracing methods should therefore produce good results. Also, the expansion schemes seem an interesting alternative to the current inversion and iterative methods. To date, the use of Z-Buffer hardware has been separated from the more realistic ray tracing methods. This need not be so, because the Z-Buffer methods essentially perform many ray queries at once and should be suitable for accelerating not just the primary rays but also particle tracing and distribution ray tracing. The

MCMC methods are the current state of the art, and so far only the simplest of these, the Metropolis method, has been used. There are many variations which could prove far better than the primitive variant presented: Prop and Wilson's [101, 135] exact sampling or Gibbs sampling [40] sound very promising. The solution to a better rendering algorithm may not lie purely with Monte Carlo methods. Instead, more integration of analytic methods such as Arvo's irradiance tensors [7, 17] seems a good idea. A hybrid scheme, whereby the type of samples used is changed from points to surfaces, would reduce noise in images and the number of samples needed.

Perceptual issues, while beyond the scope of this thesis, present many methods for highlighting redundant computation. The density estimation methods described previously are very effective at producing smooth images, but at the cost of introducing bias which replaces the noise. Silvermann suggests some multi-pass approaches which would seem to increase the accuracy of these methods. A simple enhancement to the current system is to generate PDFs during the particle tracing phase for use in the rendering phase. This scheme has been investigated by Dutre [33, 34] and Lafortune [72]. Quasi-random sequences have been shown to be very effective, but so far only the simpler sampling strategies have been implemented. In particular, $\{t, s\}$-sequences and lattices would appear to offer a lot of promise [85, 36].

Parallel approaches, even with small numbers of computers, can produce effective speedups, but not many rendering systems support more than giving each machine a different frame to render. Rarely is the scene database distributed between processors, which would enlarge the size of available memory. In relation to the schemes presented here, new path parameterisation schemes might improve coherency. Using octrees, BSP trees or Bounding Boxes or some hybrid might prove effective. Shaft culling [50] or pyramid clipping [127] could definitely be used to accelerate the traversal of the scene. Adding true scene distribution and parallel loading to the framework to decrease loading and scene preprocessing times would be beneficial.

As presented, the work in this thesis creates a state of the art rendering system which is not restricted to any particular scene type. It is capable of producing accurate view dependent and independent solutions to the rendering and potential equations. Future additions to this work will help ensure that it becomes a more capable and usable tool for solving global illumination problems.

Appendix A

Fresnel Formulae for Reflection and Refraction

These equations are derived using Maxwell's equations at a surface boundary, by making sure that energy and continuity constraints are satisfied after reflection and refraction. For the derivation, see [14]. The formulae state that the amount of light energy reflected and refracted at an interface is a function of the wavelength of the light, the geometry of the surface and the angle of incidence. The Fresnel formulae, because they use the index of refraction to calculate a materials reflective and refractive properties, are dependent on wavelength; however to simplify the formulae the index of refraction will be taken as being constant. Due to the fact that they are derived from Maxwell's equations, the Fresnel equations are written in terms of the polarisation of light. At the interface between two materials, light moves from one index of refraction N_1 to another N_2

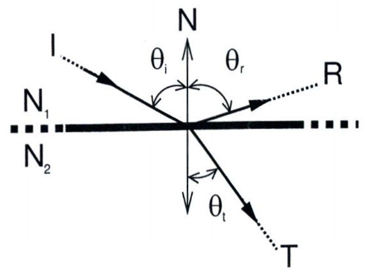


Figure A.1: Ray at an interface between mediums

(see Figure A.1). From this the *relative index of refraction* can be calculated as

$$\begin{aligned} N &= \frac{N_1}{N_2} \\ &= \frac{\eta_1 + j\kappa_1}{\eta_2 + j\kappa_2} \end{aligned}$$

Using complex arithmetic this can be expanded to

$$\begin{aligned} N &= \frac{\eta_2 + j\kappa}{\eta_1 + j\kappa_1} \frac{\eta_1 - j\kappa_1}{\eta_2 + j\kappa_2} \\ &= \frac{\eta_2\eta_1 + \kappa_2\kappa_1}{\eta_1^2 + \kappa_1^2} + j\frac{\eta_1\kappa_2 - \eta_2\kappa_1}{\eta_1^2 + \kappa_1^2} \end{aligned}$$

Using $N = \eta + j\kappa$ as the relative index of refraction, the Fresnel formula is as follows:

$$\begin{aligned} F_\perp &= \frac{a^2 + b^2 - 2a\cos\theta_i + \cos^2\theta_i}{a^2 + b^2 + 2a\cos\theta_i + \cos^2\theta_i} \\ F_\| &= F_\perp \frac{a^2 + b^2 - 2a\sin\theta_i\tan\theta_i + \sin^2\theta_i\tan^2\theta_i}{a^2 + b^2 + 2a\sin\theta_i\tan\theta_i + \sin^2\theta_i\tan^2\theta_i} \end{aligned}$$

where θ_i is the angle of incidence and a and b are given by

$$\begin{aligned} 2a^2 &= \sqrt{(\eta^2 - \kappa^2 - \sin^2\theta_i)^2 + 4\eta^2\kappa^2} + (\eta^2 + \kappa^2 - \sin^2\theta_i) \\ 2b^2 &= \sqrt{(\eta^2 - \kappa^2 - \sin^2\theta_i)^2 + 4\eta^2\kappa^2} - (\eta^2 + \kappa^2 - \sin^2\theta_i) \end{aligned}$$

The above gives the Fresnel coefficient for reflection where $F_\perp$ is the perpendicular and $F_\|$ is the parallel component of the light wave (see Figure 2.1). For unpolarised light the average of these values is used, for polarised light combinations of these value are used according to the ratio of the parallel and perpendicular vector. For transmission in a perfect medium there is no absorption, hence

$$F_r + F_t = 1 \tag{A.1}$$

To determine the irradiance we must compensate for the fact that the ray has been refracted through a medium, and thus its solid angle has been changed. This must be compensated for by

$$\text{transmission coefficient} = F_t \left(\frac{\eta_t}{\eta_i}\right)^2 \tag{A.2}$$

Note that the Fresnel functions are not BRDFs, they are used as a coefficient with the perfect BRDF.

Appendix B

Integral Equations

Image synthesis relies on the concept of integral equations, so the following appendix describes some terms, properties and definitions used in this thesis. An *integral equation* is characterised by the fact that the function to be determined b appears both inside and outside the integral. Such equations are of the form

$$b(s) = e(s) + \int_{t_1}^{t_2} \kappa(s,t)b(t)dt \tag{B.1}$$

Integral equations are categorised into two main types these are as follows:

Volterra equation have variable limits of integration:

$$b(s) = e(s) + \int_{\alpha}^{s} \kappa(s,t)b(t)dt \tag{B.2}$$

Fredholm equations have constant limits of integration:

$$b(s) = e(s) + \int_{\beta}^{\alpha} \kappa(s,t)b(t)dt \tag{B.3}$$

An integral equation is said to be of the first kind if the left hand side is equal to zero, and homogeneous if $b(s) = 0$. For each class of integral equation there are a number of methods used to solve the equation. When the boundary conditions or the equation become too complicated, numerical methods are usually used. $\kappa(s,t)$ is called the *kernel* of the integral equation and its format determines the method of solution

Symmetric : $\kappa(s,t) = \kappa(t,s)$

Convolution type : $\kappa(s,t)$ is a function of $(s-t)$

Separable : if $\kappa(s,t) = \kappa_1(s)\kappa_2(t)$

Singular : if $\kappa(s,t) \to \infty$ for some s, or when the limits are infinite.

For global illumination the class of integral equation is the Fredholm equation of the second kind as this is the form the rendering and potential equations take. This equation has the general form

$$b(s) = e(s) + \int_\beta^\alpha dt\kappa(s,t)b(t) \tag{B.4}$$

where e and the kernel κ are given, b is to be determined, and α and β are constant limits of integration. This is a Fredholm integral equation because the unknown function $b(s)$ appears outside the integral. Most integral equation properties generalise when the domain variables s and t are multi-dimensional. The above equation is abbreviated as

$$b = e + Kb \tag{B.5}$$

where K denotes the integral operator which, when applied to a function b, yields the following:

$$(Kb)(s) = \int_\beta^\alpha dt\kappa(s,t)b(t)$$

For every operator κ there exists at least another operator κ^* called the adjoint operator. The adjoint operators satisfy the following condition.

$$\langle \kappa(f), f^* \rangle = \langle \kappa^*(f^*), f \rangle \tag{B.6}$$

where $\langle f, g \rangle$ denotes the *inner product*, given by

$$\langle f, g \rangle = \int f(x)g(x)dx \tag{B.7}$$

if $\langle f, g \rangle = 0$ the functions are said to be *orthogonal.* Given a function and function adjoint to it

$$\begin{aligned} f &= a + K \\ f^* &= b + K \end{aligned}$$

we can see that

$$\int_a^b f(x)b(x)dx = \int_a^b f^*(x)a(x)dx = F \tag{B.8}$$

Thus there are a number of ways of computing F since the adjoint is not unique. The *rendering equation* and the *potential equation* described previously are examples of integral operators which are adjoint. Formally, our integral equation

B.5 can be rewritten

$$(I - K)b = e$$

where I is the identity operator.

Appendix C

Probability

This appendix covers the basic probability theory that is needed to understand some of the algorithms and concepts presented in this thesis. Most of the information presented has been adapted from [136] and [64].

C.1 Random Events

Monte Carlo methods are a numerical stochastic process, that is to say they are a sequence of random events. Random events can be divided into two categories namely *elementary* and *composite* events. Elementary events are ones which cannot be broken down into still simpler random events where as composite ones can be. For each outcome E_k of an elementary event with a countable set of outcomes there is associated a *probability* p_k such that

$$0 \leq p_k \leq 1 \cdot$$

If an outcome never occurs $p_k = 0$ and if it always occurs $p_k = 1$. Another way of expressing an events probability is

$$P(E_k) = p_k$$

The following are some properties of probabilities

1. $P\{E_i \text{ and}|\text{or } E_j\} \leq p_i + p_j$
2. E_i and E_j are said to be *mutually exclusive events* iff $E_i \implies \bar{E}_j$. If E_i and E_j are mutually exclusive then
    (a) $P\{E_i \text{ and } E_j\} = 0$
    (b) $P\{E_i \text{ or } E_j\} = p_i + p_j$

3. A whole class of events is said to be *exhaustive* if all possible events have been enumerated i.e.

$$P\{\text{ some } E_i\} = \sum_i p_i = 1$$

4. The probability of a specific outcome of two events (E_i, F_j) is called the *joint probability* for E_i and F_j.

5. Two Events E_i and F_j are said to *mutually exclusive* if $p_{ij} = p_{1i} \cdot p_{2j}$

6. Suppose E_i and F_j are not independent, then the joint probability can be written

$$\begin{aligned} p_{ij} &= \left(\sum_k p_{ik}\right)\left[\frac{p_{ij}}{\sum_k p_{ik}}\right] \\ &= p(i)\left[\frac{p_{ij}}{\sum_k p_{ik}}\right] \end{aligned} \tag{C.1}$$

   $p(i)$ is called the *marginal probability* for event E_i, that is the probability that event E_i does in fact occur, what ever the second event may be, therefore

$$\begin{aligned} \sum_i p(i) &= \sum_i \sum_k p_{ik} \\ &= 1 \end{aligned}$$

and

$$p(i) = p_{1i}.$$

   The same is true for the second event.

7. The second factor of equation C.1 is the *conditional probability*

$$p(j|i) = \frac{p_{ij}}{\sum_k p_{ik}}$$

   it is the probability for event E_j occurring given that event E_i has already occurred.

C.1.1 Discrete Random Variables

The outcome of a random event can be mapped to a numerical value. These values may have some meaningful value or be purely logical. These values x_i are associated with every elementary event E_i and are called *random variables.*

The *expectation* or *stochastic mean* value is defined as

$$\langle x\rangle = E(x) = \sum_i p_i x_i \tag{C.2}$$

The nth moment of x is defined as,

$$\begin{aligned}\langle x^n\rangle &= \sum_i p_i x_i^n \\ \langle x\rangle &= \sum_i p_i x_i = \mu\end{aligned}$$

μ is called the *expected mean value.* The *central moments* of x are given by

$$\begin{aligned}\langle g_n(x)\rangle &= \langle (x-\mu)^n\rangle \\ &= \sum_i p_i (x_i-\mu)^n\end{aligned} \tag{C.3}$$

The second central moment is called the *variance* and denoted by

$$\begin{aligned}\text{var}\{x\} &= \langle (x-\mu)^2\rangle \\ &= \sum_i p_i (x_i-\mu)^2 \\ &= \sum_i p_i x_i^2 - \langle x\rangle^2 \\ &= \langle x^2\rangle - \langle x\rangle^2\end{aligned}$$

The square root of the variance is a measure of the dispersion of the random variable. It is called the *standard deviation* or *standard error.* Another important quantity is the *covariance* which is a measure of the degree of independence of two random variables and is given by

$$\text{cov}\{x,y\} = \langle xy\rangle - \langle x\rangle\langle y\rangle$$

A quantity related to the covariance and the variance is the *correlation coefficient*

$$\rho(x,y) = \frac{\text{cov}\{x,y\}}{[\text{var}\{x\}\text{var}\{y\}]^{\frac{1}{2}}} \tag{C.4}$$

where $-1 \leq \rho \leq 1$

C.1.2 Continuous Random Variables

Continuous random variables present a small problem in that a single event or point on its own cannot have any probability but zero, thus we can only ever refer to a probability over an given region because the sum total of all probabilities must add up to one. This gives rise to *distribution functions.* Given x a real, continuous random variable,

$$-\infty < x < \infty$$

a distribution function or *cumulative distribution function* (CDF) is defined as

$$F(x) = P\{\text{a random selection of } X \text{ gives a value less than } x\} \tag{C.5}$$

This distribution function may be differentiable on certain intervals in which case a *probability density function* (PDF) may then be defined as

$$f(x) = \frac{dF(x)}{dx} \geq 0 \tag{C.6}$$

Thus the integral of this is a CDF The mean value of a continuous random variable is defined as

$$\begin{aligned} E(x) &= \int_{-\infty}^{\infty} x dF(x) \\ &= \int_{-\infty}^{\infty} x f(x) dx \end{aligned} \tag{C.7}$$

where $f(x)$ is the probability density function for x and the PDF has the normalisation property

$$\int_{-\infty}^{\infty} f(x)dx = F(\infty) = 1$$

The expectation of any function is defined as

$$E(g(x)) = \int_{-\infty}^{\infty} g(x)f(x)dx$$

Thus the moments may $E(g(x)) = \int_{-\infty}^{\infty} g(x)f(x)dx$ ıner to the discrete case

$$\langle g_n(x) \rangle = E(g^n(x)) - [E(g(x))]^n$$

From this the variance and covariance may be deduced. The joint probability may also be defined for the continuous case

$$F(x,y) = P\{X \leq x, Y \leq y\}$$

$F(x, y)$ is called a *bivariate distribution.* Again the PDF can be obtained in a similar manner to that of the discrete case as

$$f(x, y) = \frac{\partial^2 F(x, y)}{\partial x \partial y} \tag{C.8}$$

and from this the expected value of any function of x and y is given by

$$\begin{aligned} E(g(x, y)) &= \langle g(x, y) \rangle \\ &= \int f(x, y) g(x, y) dx dy \end{aligned} \tag{C.9}$$

The covariance and correlation coefficient of any continuous random variables x and y can be again defined by replacing sums with integrals. If the variables are correlated, the joint PDF as in the discrete case may be written as

$$f(x, y) = \frac{f(x, y)}{\int f(x, y) dy} \int f(x, y) dy \tag{C.10}$$

here the marginal PDF is given by

$$m(x) = \int f(x, y) dy$$

The first factor in equation C.10 is the conditional probability, that is to say given y, x may be sampled from

$$\frac{f(x|y)}{\int f(x, y) dy}$$

Equation C.10 can easily be generalised to handle more correlated random variables. In equation C.10 for a given value of y the random variable x has the conditional PDF

$$f(x|y) = \frac{f(x, y)}{\int f(x, y) dy} = \frac{f(x, y)}{m(x)}$$

The expectation of the conditional PDF called the *conditional expectation*, for fixed y is

$$\begin{aligned} E(x|y) &= \int y f(x|y) dy \\ &= \frac{\int y f(x, y) dy}{\int f(x, y') dy'} \\ &= \frac{\int y f(x, y) dy}{m(x)} \end{aligned}$$

The conditional expectation is a function of a random variable and is therefore also a random variable. Therefore an expectation of $E(x|y)$ can be found too.

Bibliography

[1] Anonymous. http://www-unix.mcs.anl.gov/mpi/. MPI Homepage.

[2] Anonymous. http://www.epm.ornl.gov/pvm/. PVM Homepage.

[3] Anonymous. MPI: a message-passing interface standard. *International Journal of Supercomputer Applications and High Performance Computing*, 8(3/4):159–416, Fall-Winter 1994.

[4] Bruno Arnauldi, Xavier Pueyo, and Josep Vilaplana. On the division of environments by virtual walls for radiosity computation. *Second Eurographics Workshop on Rendering (Photorealistic Rendering in Computer Graphics)*, pages 198–205, 1994. Held in Barcelona.

[5] J. Arvo and D.Kirk. Particle transport and image synthesis. In *Computer Graphics*, volume 24, pages 63–66. ACM Press, 1990.

[6] James Arvo. Backward ray tracing. In *SIGGRAPH '86 Developments in Ray Tracing course notes*. August 1986. also appeared in SIGGRAPH '89 Radiosity course notes.

[7] James Arvo. Applications of irradiance tensors to the simulation of non-lambertian phenomena. *Proceedings of SIGGRAPH 95*, pages 335–342, August 1995. ISBN 0-201-84776-0. Held in Los Angeles, California.

[8] Michael Ashikhmin, Simon Premoze, and Peter Shirley. A microfacet-based brdf generator. *Proceedings of SIGGRAPH 2000*, pages 65–74, July 2000. ISBN 1-58113-208-5.

[9] D. Badouel, K. Bouatouch, and T. Priol. Distributing data and control for ray tracing in parallel, 1994.

[10] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam. A user's guide to PVM: Parallel virtual machine. Technical Report ORNL/TM-11826, Mathematical Sciences Section, Oak Ridge National Laboratory, Knoxville, TN, USA, September 1991.

[11] Philippe Bekaert. http://www.cs.kuleuven.ac.be/cwis/ research/graphics/renderpark/.

[12] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. In *Communications of ACM*, volume 18, pages 509–517, 1975.

[13] James F. Blinn. Jim Blinn's corner: Fugue for MMX. *IEEE Computer Graphics and Applications*, 17(2):88–93, March/April 1997. Makes several cogent comments about deficiencies in the Intel MMX pixel-processing instruction set [95] for use in image compositing.

[14] Max Born and Emil Wolf. *Principals of Optics Sixth Edition.* Pergamon Press, Oxford 0X3 0BW, England, 1993.

[15] Kadi Bouatouch, S. N. Pattanaik, and Eric Zeghers. Computation of higher order illumination with a non-deterministic approach. *Computer Graphics Forum*, 15(3):327–338, August 1996. ISSN 1067-7055.

[16] Alan Chalmers and Erik Reinhard. *SIGGRAPH 98 "Parallel and Distributed Photo-Realistic Rendering" course #3 notes.* July 1998.

[17] Min Chen and James Arvo. A closed-form solution for the irradiance due to linearly-varying luminaires. *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 137–148, June 2000. ISBN 3-211-83535-0.

[18] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics (Proceedings of SIGGRAPH 88)*, 22(4):75–84, August 1988. Held in Atlanta, Georgia.

[19] Michael F. Cohen and John R. Wallace. Radiosity and realistic image synthesis. 1993. Held in San Diego, CA.

[20] Steven Collins. Adaptive Splatting for Specular to Diffuse Light Transport. In *Fifth Eurographics Workshop on Rendering*, pages 119–135, Darmstadt, Germany, June 1994.

[21] Steven Collins. *Wavefront Tracking for Global Illumination Solutions.* Ph.D. thesis, Dept. of Computer Science, Trinity College Dublin, January 1997.

[22] Steven Collins. Adaptive splatting for specular to diffuse light transport. *Fifth Eurographics Workshop on Rendering*, pages 119–135, June 1994. Held in Darmstadt, Germany.

[23] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The reyes image rendering architecture. *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 95–102, July 1987. Held in Anaheim, California.

[24] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):137–145, July 1984. Held in Minneapolis, Minnesota.

[25] Antonio Corradi, Letizia Leonardi, and Franco Zambonelli. Diffusive load-balancing policies for dynamic applications. *IEEE Concurrency*, 7(1), January/March 1999.

[26] B. J. Cox. There is a silver bullet: The birth of interchangeable, reusable software components will bring software into the information age. *Byte Magazine*, 15(10):209–210, 212, 214, 216, 218, October 1990.

[27] Phil Daley. Run length encoding revisited. *Dr. Dobb's Journal of Software Tools*, 14(5):130, 154, May 1989.

[28] L.P. Deutsch. Deflate compressed data format specification. www.info-zip.org/pub/infozip/zlib/rfc-deflate.html.

[29] L.P. Deutsch. Gzip compressed data format specification. www.info-zip.org/pub/infozip/zlib/rfc-gzip.html.

[30] Keith Diefendorff, Pradeep K. Dubey, Ron Hochsprung, and Hunter Scales. AltiVec extension to PowerPC accelerates media processing. *IEEE Micro*, 20(2):85–95, March/April 2000.

[31] P. Dutre, E. Lafortune, and Y. D. Willems. Monte Carlo Light Tracing with Direct Pixel Contributions. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 128–137, Alvor, Portugal, December 1993.

[32] Ph. Dutre, E. Lafortune, and Y. D. Willems. A mathematical framework for global illumination algorithms. *Winter School of Computer Graphics 1994*, January 1994. Held in held at University of West Bohemia, Plzen, Czech Republic, 19-20 January 1994.

[33] Philip Dutré and Yves D. Willems. Importance-driven monte carlo light tracing. In Georgios Sakas and Müller [41], pages 188–200. Proc. 5th Eurographics Rendering Workshop, Darmstadt,Germany, June 13–15, 1994.

[34] Philip Dutré and Yves D. Willems. Potential-driven monte carlo particle tracing for diffuse environments with adaptive probability density functions. *Eurographics Rendering Workshop 1995*, pages 306–315, June 1995. Held in Dublin, Ireland.

[35] Frank Suykens Yves D.Willems. Adaptive filtering for progressive monte carlo image rendering. *Proceedings of WSCG 2000*, 2000. Held in Plzen, Czech Republic.

[36] Michael Evans and Tim Swartz. *Approximating Integrals vis Monte Carlo and Deterministic Methods.* Oxford University Press, Great Clarendon Street,Oxford OX2 6DP, England, 2000.

[37] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Computers*, C-21(9):948–960, September 1972.

[38] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. ARTS: Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986.

[39] G.A. Geist, J.A. Kohl, and P.M. Papadopoulos. Mpi and pvm: A comparison of features. *Calculateurs Paralleles*, 8(2), May 1996. Describes differences between PVM and MPI and under what circumstances one package is favored over the other.

[40] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions of Pattern Recognition and Machine Intelligence*, 6:721–741, November 1984.

[41] Peter Shirley Georgios Sakas and Stefan Müller, editors. *Photorealistic Rendering Techniques*, Eurographics. Springer-Verlag Berlin Heidelberg New York, 1994. Proc. 5th Eurographics Rendering Workshop, Darmstadt,Germany, June 13–15, 1994.

[42] Andrew Glassner. Space subdivision for fast raytracing. *IEEE Computer Graphics and Applications*, 6:15–22, 1984.

[43] Andrew Glassner. Spectrum: a proposed image synthesis architecture. *SIGGRAPH '91 Frontiers in Rendering course notes*, July 1991.

[44] Jay S. Gondek, Gary W. Meyer, and Jonathan G. Newman. Wavelength dependent reflectance functions. *Proceedings of SIGGRAPH 94*, pages 213–220, July 1994. ISBN 0-89791-667-0. Held in Orlando, Florida.

[45] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modelling the interaction of light between diffuse surfaces. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):213–222, July 1984. Held in Minneapolis, Minnesota.

[46] Cindy M. Goral, Kenneth E. Torrance, and Donald P.Greenberg. http://www.graphics.cornell.edu/online/box/.

[47] Steven J. Gortler, Michael F. Cohen, and Phillip Slusallek. Radiosity and relaxation methods: Progressive refinement is southwell relaxation. February 1993.

[48] Stuart A. Green and Derek J. Paddon. Exploiting coherence for multiprocessor ray tracing. *IEEE Computer Graphics and Applications*, 9(6):12–26, November 1989.

[49] J. L. Gustafson. Reevaluating amdahl's law. *Commun. of the ACM*, 31, 5:532–533, 1988.

[50] Eric A. Haines and John R. Wallace. Shaft culling for efficient ray-traced radiosity. *Second Eurographics Workshop on Rendering (Photorealistic Rendering in Computer Graphics)*, 1994. Held in Barcelona.

[51] Patrick M. Hanrahan and Werner Purgathofer, editors. *Rendering Techniques '95*, Eurographics. Springer-Verlag Wien New York, 1995. Proc. 6th Eurographics Rendering Workshop, Dublin,Ireland, June 12–14, 1995.

[52] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):145–154, August 1990. ISBN 0-201-50933-4. Held in Dallas, Texas.

[53] S. Heinrich and Alexander Keller. Quasi-Monte Carlo Methods in Computer Graphics Part I: The QMC-Buffer. Technical Report 242/94, University of Kaiserlautern, 1994.

[54] S. Heinrich and Alexander Keller. Quasi-Monte Carlo Methods in Computer Graphics Part II: The Radiance Equation. Technical Report 243/94, University of Kaiserlautern, 1994.

[55] Alan Heirich and James Arvo. Scalable monte carlo image synthesis. *Parallel Computing*, 23(7):845–859, July 1997.

[56] Michi Henning, Steve Vinoski, and Stephen Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999.

[57] W. D. Hillis and G. L. Steele. *The Connection Machine*. M.I.T. Press, 1986.

[58] Frederik W. Jansen and Erik Reinhard. Data locality in parallel rendering. In *Second Eurographics Workshop on Parallel Graphics and Visualisation*, pages 1–15. Eurographics, September 1998.

[59] Henrik Wann Jensen. Importance-driven path tracing using the photon map. In Hanrahan and Purgathofer [51], pages 326–335. Proc. 6th Eurographics Rendering Workshop, Dublin,Ireland, June 12–14, 1995.

[60] Henrik Wann Jensen. Rendering caustics on non-lambertian surfaces. *Computer Graphics Forum*, 16(1):57–64, 1997. ISSN 0167-7055.

[61] Henrik Wann Jensen. Global illumination using photon maps. *Eurographics Rendering Workshop 1996*, pages 21–30, June 1996. ISBN 3-211-82883-4. Held in Porto, Portugal.

[62] Henrik Wann Jensen and Niels Jørgen Christensen. Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers & Graphics*, 19(2):215–224, March 1995. ISSN 0097-8493.

[63] James T. Kajiya. The rendering equation. *Computer Graphics (Proceedings of SIGGRAPH 86)*, 20(4):143–150, August 1986. Held in Dallas, Texas.

[64] Malvin H. Kalos and Paula A. Whitlock. *Basics*, volume 1 of *Monte Carlo Methods*. John Wiley and Sons, New York, Chichester, Brisbane, Toronto and Singapore, 1986.

[65] Alexander Keller. The Fast Calculation of Form Factors Using Low Discrepancy Sequences. In *Proceedings of the Spring Conference on Computer Graphics (SCCG '96)*, pages 195–204, Bratislava, Slovakia, June 1996. Comenius University Press. Available from http://www.dcs.fmph.uniba.sk/ sccg/proceedings/1996.index.htm.

[66] Alexander Keller. Quasi-monte carlo radiosity. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 101–110, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

[67] David Kirk and James Arvo. The ray tracing kernel. *Proceedings of Ausgraph '88*, pages 75–82, 1988.

[68] David J. Kruglinski. *Inside Visual C++*. Microsoft Press, Redmond, Washington, fourth edition, 1997.

[69] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *CompuGraphics*, pages 145–153, 1993.

[70] Eric P. Lafortune and Yves D. Willems. The ambient term as a variance reducing technique for monte carlo ray tracing. In Georgios Sakas and Müller [41], pages 168–176. Proc. 5th Eurographics Rendering Workshop, Darmstadt,Germany, June 13–15, 1994.

[71] Eric P. Lafortune and Yves D. Willems. Using the modified phong brdf for physically based rendering. Technical Report CW197, Department of Computer Science, K.U.Leuven, November 1994. This text discusses a few aspects of reflectance models in physically based rendering.

[72] Eric P. Lafortune and Yves D. Willems. A 5d tree to reduce the variance of monte carlo ray tracing. In Hanrahan and Purgathofer [51], pages 11–20. Proc. 6th Eurographics Rendering Workshop, Dublin,Ireland, June 12–14, 1995.

[73] Eric P. Lafortune and Yves D. Willems. Rendering participating media with bidirectional path tracing. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 91–100, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

[74] Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Non-linear approximation of reflectance functions. *Proceedings of SIGGRAPH 97*, pages 117–126, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

[75] Paul Lalonde and Alain Fournier. A wavelet representation of reflectance functions. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):329–336, October - December 1997. ISSN 1077-2626.

[76] Gregory Ward Larson, Holly Rushmeier, and Christine Piatko. A visibility matching tone reproduction operator for high dynamic range scenes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):291–306, October - December 1997. ISSN 1077-2626.

[77] William Leeson and Carol O'Sullivan. Effigi:an efficient framework for implementing global illumination. In *Proceedings of the 8-th International Conference in Central Europe on Computer Graphics, Visualisation and Interactive Digital Media'2000*, pages 62–69, Plzen, Czech Republic.

[78] William Leeson and Carol O'Sullivan. Metropolis particle tracing for global illumination. In *Proceedings of the IASTED International Conference on Computer Graphics and Imaging*, pages 51–56, Las Vegas, Nevada, USA.

[79] G. P. LePage. A new algorithm for adaptive multidimensional integration. In *Journal of Computational Physics*, volume 27, pages 192–203, 1978.

[80] G. P. Lepage. Vegas: An adaptive multidimensional integration program. Technical Report CLNS-80/447, Cornell University, 1980.

[81] Jean loup Gailly and Mark Adler. http://www.info-zip.org/pub/infozip/zlib/.

[82] Don P. Mitchell. Consequences of stratified sampling in graphics. *Proceedings of SIGGRAPH 96*, pages 277–280, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[83] László Neumann, Attila Neumann, and László Szirmay-Kalos. Compact metallic reflectance models. *Computer Graphics Forum*, 18(3):161–172, September 1999. ISSN 1067-7055.

[84] László Neumann, Attila Neumann, and László Szirmay-Kalos. Reflectance models with fast importance sampling. *Computer Graphics Forum*, 18(4):249–265, December 1999. ISSN 1067-7055.

[85] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* SIAM, 1992.

[86] Tomoyuki Nishita and Eihachiro Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. *Computer Graphics (Proceedings of SIGGRAPH 85)*, 19(3):23–30, July 1985. Held in San Francisco, California.

[87] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smiths, and Charles Hansen. Interactive ray tracing.

[88] Steven Parker, Peter-Pike Sloan, Peter Shirley, Brian Smiths, Yarden Livnat, and Charles Hansen. Interactive ray tracing for isosurface rendering.

[89] S. Pattanaik. The mathematical framework of adjoint equations for illumination computations. *Proceedings of ICCG*, pages 265–288, February 1993. Held in Bombay, India.

[90] S. Pattanaik and S. Mudur. Efficient potential equation solutions for global illumination computation. *Computers & Graphics*, 17(4):387–396, 1993. ISSN 0097-8493.

[91] S. N. Pattanaik and S. P. Mudur. Computation of global illumination in a participating medium by monte carlo simulation. *The Journal of Visualization and Computer Animation*, 4(3):133–152, July - September 1993.

[92] S. N. Pattanaik and S. P. Mudur. The potential equation and importance in illumination computations. *Computer Graphics Forum*, 12(2):131–136, 1993. Held in Cambridge, UK.

[93] S. N. Pattanaik and S. P. Mudur. Adjoint equations and random walks for illumination computation. *ACM Transactions on Graphics*, 14(1):77–102, January 1995. ISSN 0730-0301.

[94] S. N. Pattanaik and S. P. Mudur. Computation of global illumination by monte carlo simulation of the particle model of light. *Third Eurographics Workshop on Rendering*, pages 71–83, May 1992. Held in Bristol, UK.

[95] Alex Peleg, Sam Wilkie, and Uri Weiser. Intel MMX for multimedia PCs. *Communications of the ACM*, 40(1):24–38, January 1997. See also Blinn's comments [13] about MMX instruction set deficiencies.

[96] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. *Proceedings of SIGGRAPH 97*, pages 101–108, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

[97] Bui-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311—317, June 1975.

[98] Pierre Poulin. Separate functions for local illumination. *Graphics Interface '92 Workshop on Local Illumination*, pages 37–43, May 1992.

[99] W.H. Press and G.R. Farrar. In *Computers in Physics*, volume 4, pages 190–195, 1990.

[100] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, 2 edition, 1992.

[101] James Gary Propp and David Bruce Wilson. How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *Journal of Algorithms*, 27(2):170–217, May 1998.

[102] Richard A. Redner, Mark E. Lee, and Samuel P. Uselton. Smooth B-spline illumination maps for bidirectional ray tracing. *ACM Transactions on Graphics*, 14(4):337–362, October 1995. Corrections to Figures 4–9 are available on the World-Wide Web at http://www.acm.org/tog/AandE.html.

[103] Erik Reinhard. Hybrid scheduling for parallel ray tracing. Technical report, Faculty of Technical Mathematics and Informatics, Delft University of Technology, January 1996.

[104] Erik Reinhard. *Scheduling and Data Management for Parallel Ray Tracing.* PhD thesis, Department of Computer Science, University of Bristol, October 1999.

[105] Erik Reinhard, Alan Chalmers, and Frederik W Jansen. Hybrid scheduling for parallel rendering using coherent ray tasks. In *1999 IEEE Parallel Visualization and Graphics Symposium*, pages 21–28. ACM SIGGRAPH, October 1999.

[106] Erik Reinhard and Frederik W. Jansen. Hybrid scheduling for efficient ray tracing of complex images. In *High Performance Computing for Computer Graphics and Visualisation*, pages 78–87. Springer-Verlag, July 1995.

[107] Erik Reinhard and Frederik W Jansen. Scheduling issues in parallel rendering. In *Proceedings of the First Annual Conference of the Advanced School for Computing and Imaging*, pages 268–277. ASCI, May 1995.

[108] Erik Reinhard and Frederik W. Jansen. Rendering large scenes using parallel ray tracing. In *First Eurographics Workshop on Parallel Graphics and Visualisation*, pages 67–80. Alpha Books, September 1996.

[109] Erik Reinhard, Arjan J. F. Kok, and Frederik W. Jansen. Cost prediction in ray tracing. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 41–50, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

[110] Dale Rogerson. *Inside COM.* Microsoft Press, New York, 1997.

[111] François Rousselle and Christophe Renaud. Group accelerated shooting methods for radiosity. *Eurographics Rendering Workshop 1999*, June 1999. Held in Granada, Spain.

[112] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. volume 14, pages 110–116, July 1980.

[113] Christophe Schlick. A survey of shading and reflectance models. *Computer Graphics Forum*, 13(2):121–131, June 1994.

[114] Peter Shirley. *Physically Based Lighting Calculations for Computer Graphics.* Ph.D. thesis, Dept. of Computer Science, U. of Illinois, Urbana-Champaign, November 1990. good overview of illumination algorithms to date.

[115] Peter Shirley. Nonuniform Random Point Sets via Warping. In David Kirk, editor, *Graphics Gems III*, pages 80–83. Academic Press Professional, Boston, MA, 1992.

[116] Peter Shirley, Helen Hu, Brian Smits, and Eric P. Lafortune. A practitioners' assessment of light reflection models. *Pacific Graphics '97*, October 1997. Held in Seoul, Korea.

[117] Peter Shirley, Kelvin Sung, and William Brown. A ray tracing framework for global illumination systems. *Graphics Interface '91*, pages 117–128, June 1991.

[118] Peter Shirley and Chang Yaw Wang. Distribution ray tracing: Theory and practice. *Third Eurographics Workshop on Rendering*, pages 33–43, May 1992. Held in Bristol, UK.

[119] Peter Shirley, Chang Yaw Wang, and Kurt Zimmerman. Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, January 1996. ISSN 0730-0301.

[120] Francios X. Sillion and Jean-Marc Hasenfratz. Efficient parallel refinement for hierarchical radiosity on a dsm computer. In *Proceedings of Third Eurographics Workshop on Parallel Graphics and Visualisation*, Girona, Spain, 2000.

[121] B.W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, 11 New Fetter Lane, London EC4P 4EE, 1986.

[122] P. Slusallek and Hans-Peter Siedel. Vision - an architecture for global illumination calculations. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):77–96, March 1995. ISSN 1077-2626.

[123] László Szirmay-Kalos. Stochastic iteration for non-diffuse global illumination. *Computer Graphics Forum*, 18(3):233–244, September 1999. ISSN 1067-7055.

[124] B. Trumbore, W. Lytle, and D. P. Greenberg. A testbed for image synthesis. *Eurographics '91*, pages 467—480, September 1991.

[125] Jack Tumblin and Holly E. Rushmeier. Tone reproduction for realistic images. *IEEE Computer Graphics & Applications*, 13(6):42–48, November 1993.

[126] Maurice van der Zwaan, Erik Reinhard, and Frederik W. Jansen. Pyramid Clipping for Efficient Ray Transversal. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 1–10, New York, NY, 1995. Springer-Verlag.

[127] Maurice van der Zwaan, Erik Reinhard, and Frederik W. Jansen. Pyramid clipping for efficient ray traversal. *Eurographics Rendering Workshop 1995*, pages 1–10, June 1995. Held in Dublin, Ireland.

[128] Eric Veach. Non-symmetric scattering in light transport algorithms. *Eurographics Rendering Workshop 1996*, pages 81–90, June 1996. ISBN 3-211-82883-4. Held in Porto, Portugal.

[129] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. *Fifth Eurographics Workshop on Rendering*, pages 147–162, June 1994. Held in Darmstadt, Germany.

[130] Eric Veach and Leonidas J. Guibas. Metropolis light transport. *Proceedings of SIGGRAPH 97*, pages 65–76, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

[131] Bruce Walter, Philip M. Hubbard, Peter Shirley, and Donald F. Greenberg. Global illumination using local linear density estimation. *ACM Transactions on Graphics*, 16(3):217–259, July 1997. ISSN 0730-0301.

[132] Bruce Jonathon Walter. *Density Estimation Techniques For Global Illumination.* Ph.D. thesis, Dept. of Computer Science, Cornell University, August 1998.

[133] Changyaw Wang. Physically correct direct lighting for distribution ray tracing. In David Kirk, editor, *Graphics Gems III*, pages 307–313, 562–568. Academic Press, Boston, MA, 1992.

[134] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics (Proceedings of SIGGRAPH 88)*, 22(4):85–92, August 1988. Held in Atlanta, Georgia.

[135] David Bruce Wilson and James Gary Propp. How to get an exact sample from a generic markov chain and sample a random spanning tree from a directed graph, both within the cover time. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, New York/Philadelphia, January 28–30 1996. ACM/SIAM.

[136] Reuven Y.Rubinstein. *Simulation and the Monte Carlo Method.* John Wiley and Sons, New York, 1981.

[137] Harold R. Zatz. Galerkin radiosity: A higher order solution method for global illumination. *Proceedings of SIGGRAPH 93*, pages 213–220, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California.

[138] Robert Zigon. Run length encoding. *Dr. Dobb's Journal of Software Tools*, 14(2), February 1989.

Index