Title
A Comprehensive Study of the Seminal Monte-Carlo-Based Image Synthesis Algorithms

Author
Kouhzadi, Ali

Publication Date
2016

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

A Comprehensive Study of the Seminal Monte-Carlo-Based Image Synthesis Algorithms

A thesis submitted in partial satisfaction
of the requirements for the degree Master of Science
in Computer Science

by

Ali Kouhzadi

2016

ABSTRACT OF THE THESIS

A Comprehensive Study of the Seminal
Monte-Carlo-Based Image Synthesis Algorithms

by

Ali Kouhzadi

Master of Science in Computer Science
University of California, Los Angeles, 2016
Professor Demetri Terzopoulos, Chair

This thesis presents an in-depth study of the problem of photorealistic image synthesis in the field of computer graphics. To better understand the proposed solutions to this problem, we first briefly overview several relevant fundamental topics, such as probability, Monte Carlo integration, sampling methods, reflection methods, etc. We then review some of the most influential Monte-Carlo-Based image synthesis methods introduced over the past four decades. Finally, to better understand the advantages and weaknesses of each method, we evaluate the reviewed methods using the Mitsuba Renderer, an open-source, research-oriented rendering system.

The thesis of Ali Kouhzadi is approved.

Song-Chun Zhu

Joseph M. Teran

Demetri Terzopoulos, Committee Chair

University of California, Los Angeles

2016

Contents

List of Figures

List of Tables

1. Introduction

1.1 Global Illumination and Photorealistic Image Synthesis

Global Illumination refers to a group of algorithms concerned with simulating the interaction of light with surfaces and volumes [10, p. 470], also called light transport simulation. The purpose of global illumination is in synthesizing photorealistic images based on a scene description. In the context of computer graphics, geometrical optics is almost exclusively used for simulating the behavior of light [13, 20] and other theoretical frameworks (e.g., light as waves or particles) are usually ignored, since they are either too complex and negatively affect efficiency, or do not have noticeable visual effects at macroscopic scales.

The first attempt at approaching image synthesis using geometrical optics can be traced back to Appel's 1968 paper [2], which laid the foundations for Goldstein and Nagel's 1971 milestone paper that introduced the first method for simulating the physical process of photography and synthesizing fully computer-generated perspective images by "geometric ray tracing". These two seminal papers paved the way for the introduction of recursive ray tracing by Whitted in 1979 [36], ray casting by Roth in 1982 [28], and radiosity by Goral et al. in 1984 [9]. In turn, the introduction of Whitted's recursive ray tracing and Goral's radiosity were arguably the start of a wave of scholarly interest in global illumination and physically-based image synthesis.

As was discussed in [2], the most obvious method for simulating light transport is to trace rays emitted from a light source through the scene until they reach the observer, but in practice this method is not feasible since it is most likely that a small number, or even none, of the selected rays reach the observer (e.g., in outdoor scenes). Tracing infinite beams of

light emitted from a light source is also impractical for obvious reasons. The more practical alternative is to trace light rays or beams of light backward, from the observer through the image plane, and from there through the scene until some of them reach some sort of light source. This is essentially the basis of all the global illumination methods in computer graphics.

There are two types of widely-used light transport models:

1. Local Illumination and

2. Global Illumination.

In the local illumination approach, only direct lighting is taken into account—light rays that are directly emitted from the light source and bounce off a single surface directly towards the observer [10, p. 140]. Using this approach, all the inter-reflections between different surfaces in a scene are ignored in exchange for performance, which makes it well-suited for realtime application (e.g., video games, virtual reality). In realtime rendering, a local lighting model is used by default, and numerous "hacks and tricks" and ad-hoc methods are used to efficiently synthesize natural-looking images.

Global illumination methods, by contrast, take into account direct lighting as well as indirect lighting, so that the process of shading a point on a surface takes into account the direct light rays coming from the light sources, the reflected light rays coming from other objects in the scene (inter-reflection), and refracted light rays passing through the object itself, as well as the effects of participating media, etc., thus producing photorealistic images. Using global illumination, one can simulate optical phenomena such as caustics, color-bleeding, soft shadows, motion blur, etc.

Subsequent improvements on recursive ray tracing and radiosity resulted in one of the most important contributions to the field in 1986 by Kajiya—the Rendering Equation, also referred to as the Light Transport Equation (LTE), which was introduced in his seminal paper [20] and was the starting point of Monte Carlo Ray Tracing (MCRT), methods that

involve evaluating the rendering equation using Monte Carlo integration. These methods can be categorized into two groups of algorithms (to be discussed in more detail in Chapter 2):

1. Biased and

2. Unbiased.

In general, we can categorize global illumination algorithms into two groups:

1. Algorithms based on point sampling and Monte Carlo integration (e.g., path tracing, bidirectional path tracing, Metropolis light transport, etc.)

2. Algorithms based on the Finite Element Method (FEM) (e.g., radiosity)

Over time and through the introduction of new methods, it is now widely accepted that the most general and robust approach to the global illumination problem is MCRT [13, p. vi].

1.2 Objectives

The main goal of this thesis is to conduct a focused and in-depth study of the topic of photorealistic image synthesis and physically-based rendering.

Based on research that has been presented at the ACM SIGGRAPH Conference in recent years, the film and animation industry seems to be moving toward replacing the decades old, industry-standard rendering techniques based on rasterization and ad-hoc methods by Monte Carlo Ray Tracing methods. Moreover, hardware manufacturers such as NVIDIA and Intel corporations show a high level of interest, at least in principle, in a not so far off future when GPU architectures based on ray-tracing could replace the current industry-standard GPU architectures based solely on rasterization. Last but not least, the big players in the video game industry more or less show the same level of enthusiasm and optimism in achieving photorealism using global illumination and ray-tracing methods, a great example being the

advent of game engines like Unreal Engine 4[1] in recent years that are capable of taking advantage of ray-tracing to render soft shadows.

Based on the above considerations, now seems to be the perfect time to study this particular subfield of Computer Graphics and understand the intricacies of these methods. This thesis reports on our investigation into the most prominent existing methods for solving the rendering equation.

1.3 Outline

Chapter 2 covers the prerequisite mathematical topics needed to understand the Monte-Carlo-based rendering techniques, including basic probability concepts, Monte Carlo integration, geometrical optics, light scattering, etc.

Chapter 3 discusses the seminal MCRT methods in chronological order, starting with a brief discussion and introduction of precursors to MCRT, namely ray-casting, ray-tracing, and its variations, as well as radiosity, and then moving on to advanced Monte Carlo rendering methods, culminating in the Stochastic Progressive Photon Mapping algorithm introduced in 2009.

Chapter 4 presents a quantitative comparison of the introduced methods and sample images rendered using the Mitsuba open source renderer,[2] along with a discussion of the comparative advantages and disadvantages of these methods.

Chapter 5 presents our conclusions, as well as a discussion of the opportunities for improving global illumination methods, along with providing pointers to advanced topics such as texture mapping, volumetric rendering, etc.

[1] Epic Games, Inc.
[2] Website: https://www.mitsuba-renderer.org/

2. Background

2.1 Basic Probability Concepts

As a precursor to discussing Monte Carlo integration, we review in this section basic probability concepts that should be covered in an undergraduate probability and statistics course. The material in this section is based on [27, Chapter 13] and [22, Chapter 1]. Readers unfamiliar with basic probability concepts are encouraged to refer to these two references for more detailed discussions.

A continuous random variable x is a scalar or vector quantity that randomly takes on some value from the real line $R = (-\infty, +\infty)$. The behavior of x is described by the distribution of values that it takes using a Probability Density Function, or pdf for short; x p denotes the relationship between a pdf p associated with a random variable x.

The probability that x will take on a value in interval $[a, b]$ is given by

$$\textit{Probability}(x \in [a, b]) = \int_a^b p(x)dx. \tag{2.1}$$

A pdf (here denoted as function p) describes the relative likelihood of a random variable taking a certain value. For instance, if $p(x_1) = 0.6$ and $p(x_2) = 0.3$, then a random variable with density p is twice as likely to have a value near x_1 than near x_2.

Density p has the following characteristics:

- $p(x) \geq 0$;
- $\int_{-\infty}^{+\infty} p(x)dx = 1$.

The average value that a real function f of a one-dimensional random variable with pdf

p will take on is called its expected value or mean, denoted by $E[f(x)]$ and obtained by

$$E[f(x)] = \int_{-\infty}^{+\infty} f(x)p(x)dx. \tag{2.2}$$

The expected value of the function f has the following properties:

- $E[f(x) + g(y)] = E[f(x)] + E[g(y)]$;
- $E[a.f(x) + b] = a.E[f(x)] + b$.

A measure of the amount of variation of the values a random variable might take is described in terms of its variance, showing how far the data are spread out relative to the mean. For one-dimensional random variable, variance is obtained as

$$V(x) = E[(x - E[x])^2]. \tag{2.3}$$

Note that it can be proven (2.3) can also be written as

$$V(x) = E[x^2] - (E[x])^2. \tag{2.4}$$

The standard deviation or σ describes the distance (difference) between a value and the mean, it is also called the standard error of a sampling distribution. Using standard deviation one can investigate the following:

1. How much variation exists from the mean?
2. How are the data spread on a specific interval?
3. The expected difference between an individual measurement and the mean.

The standard deviation is simply the square root of the variance:

$$\sigma = \sqrt{V(x)}. \tag{2.5}$$

Random variables and expected value can also be defined in two or more dimensions. Let X and Y be two continuous random variables, and $f(x, y)$ be their joint probability density function (joint pdf); the probability that X and Y take on values in any two-dimensional set A is the volume under the density surface above A

$$P[(X, Y) \in A] = \iint_A f(x, y)dxdy. \tag{2.6}$$

Suppose the space S has associated measure μ; for instance, S is the surface of a sphere and μ measures its area. We can define a pdf $p : S \to R$, and if x is a random variable with x p, then the probability that x will take on a value in some region $S_i \subset S$ is given by the integral

$$Probability(x \in S_i] = \int_{S_i} p(x)d\mu. \tag{2.7}$$

Equation (2.7) gives the probability that x takes on a value in the region S_i. In computer graphics, S is often an area (A), therefore $d\mu = dA = dx \times dy$, or a set of directions (e.g., points on a unit sphere); therefore, $d\mu = d\omega = \sin\theta \times d\theta \times d\varphi$.

In two dimensions, the expected value of the continuous random variable X is

$$E[X] = \mu_X = \iint_{-\infty}^{+\infty} xf(x, y)dxdy. \tag{2.8}$$

And for Y,

$$E[Y] = \mu_Y = \iint_{-\infty}^{+\infty} yf(x, y)dxdy. \tag{2.9}$$

And, finally, the expected value of a function of X and Y, $g(X, Y)$ is

$$E[g(X, Y)] = \mu_g = \iint_{-\infty}^{+\infty} g(x, y)f(x, y)dxdy. \tag{2.10}$$

2.2 Estimators and Their Properties

Another concept that must be discussed before the introduction of Monte Carlo Integration is how to estimate the expected value of a random variable. Summation of independent, identically distributed random variables x_i, independent random variables that share a common pdf p, divided by the number of random variables gives an estimate for the $E[x]$:

$$E[X] \approx \frac{1}{N} \sum_{i=1}^{N} x_i. \tag{2.11}$$

As N increases, the variance of (2.11) decreases; therefore, a large enough N is desired in order to decrease the error of the estimate.

A function F_N of N random variables is called an estimator for a quantity Q, if its expected value can be used as an estimate to Q, meaning $Q \approx E[F_N]$.

The error of an estimator F_N is its difference from the exact value of Q:

$$error = F_N - Q. \tag{2.12}$$

To make it more general, a more useful definition is the Mean Square Error of an estimator, defined as

$$MSE = E[(F_N - Q)^2]. \tag{2.13}$$

The bias of an estimator is defined as the expected value of its error:

$$\beta[F_N] = E[F_N - Q]. \tag{2.14}$$

An estimator F_N is called unbiased if $\beta[F_N] = 0$, which in turn indicates

$$E[F_N - Q] = 0 \Rightarrow E[F_N] = Q. \tag{2.15}$$

The advantage of an unbiased estimator is that it is guaranteed to produce correct results given a large enough number of samples. The variance of an unbiased estimator is the same as its MSE. On the other hand, a biased estimator does not guarantee to produce correct results, regardless of number of samples taken, and its variance is not the same as its MSE.

In the context of unbiased MCRT, variance (or MSE) manifests itself as noisy pixels in the output image; however, a biased MCRT method can produce images with less noise, although results are not guaranteed to be correct.

Another important concept introduced in [22, Chapter 1] is the concept of consistency of an estimator; an estimator is said to be consistent if its error converges to zero as the number of samples increases. Therefore, an unbiased estimator that is not consistent can produce incorrect results, while a biased estimator that is consistent can produce correct results.

To summarize, unbiased MCRT methods, as opposed to biased methods, do not contain any systematic error—they can be stopped after any number of samples and the expected value of the estimator will be correct regardless. The biggest problem with unbiased algorithms is their variance, which will show up in the rendered image as noise that can be reduced by taking more samples. Biased algorithms can also produce correct results, but they can converge to the correct result by taking more samples [13]. In computer graphics, the first unbiased Monte Carlo method was introduced by Kajiya [20], which was based on earlier works by Cook et al. (1984) and Whitted (1980) [33].

Examples of unbiased MCRT methods are Path Tracing [20], Bi-directional Path Tracing [23, 24, 31, 32], and Metropolis Light Transport [33]. Irradiance Caching [35], Density Estimation [30], and Photon Mapping [15] are examples of biased MCRT methods.

2.3 Monte Carlo Integration

The Monte Carlo Integration method is a numerical method for approximating complex integrals. Given a function $f : S \to R$ and a random variable x p, the expected value of

$f(x)$ is approximated by

$$E[f(x)] = \int_{x \in S} f(x)p(x)d\mu \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{2.16}$$

We can take advantage of (2.16) to approximate any integral by an appropriate sum. To simplify (2.16) and make it more general, we make the assumption that $g = f \times p$, where p is always positive and $g \neq 0$, giving us the following equation:

$$\int_{x \in S} g(x)d\mu \approx \frac{1}{N}\sum_{i=1}^{N} \frac{g(x_i)}{p(x_i)}. \tag{2.17}$$

To make estimations better in general, we should try to

1. Take as many samples as possible.

2. g and p should have a similar shape; in other words, $\frac{g}{p}$ should have a low variance.

For a more detailed discussion on Monte Carlo integration, refer to [27, Chapters 13, 14]

2.4 Geometrical Optics and Light Rays

In the context of computer graphics, we are mostly interested in simulating the interaction of light with matter using geometrical optics. The true nature of light according to Quantum Mechanics, or whether light acts exactly as particles or waves in a specific situation are not of concern here, although these topics (electromagnetic waves and photons) will be touched very briefly in the correct context (e.g., radiometry).

It is useful to think of light traveling as hypothetical constructs such as rays and beams [34]. A Light Beam is a very thin collection of infinitesimal Light Rays traveling in the same direction, or one can think of it as the path in which photons travel from the light source in some specific direction. Although a light beam usually refers to a collection of parallel light rays, in the context of rendering they are sometimes used interchangeably.

Two basic rules describing light rays are [34]:

1. In a single unchanging medium, they travel in straight lines.

2. They are reversible, meaning that one can trace their path in opposite direction.

Studying light in terms of these hypothetical constructs helps us simplify the problem while giving an accurate approximation of the behavior of light in the real world; for instance, light scattering, reflection, and refraction.

2.5 Heckbert's Light Transport Notation

A notation to describe a light transport path is desirable in order to avoid verbose descriptions of a path bouncing off of several surfaces. Often in rendering literature a notation known as Heckbert's notation is used to describe light transport paths and different types of reflections at each bouncing point using regular expressions. This method was first introduced in [12]. The following symbols are used to describe different types of light-surface interactions along a light transport path:

- E and L are two ends of a path—eye and light respectively.
- D is the diffuse reflection or transmission.
- G is the glossy reflection or transmission.
- S is the specular reflection or refraction.

2.6 Radiometry

Radiometry is concerned with the measurement of electromagnetic radiation as well as describing light propagation and reflection [26, p. 202][27, p 281]. It forms the basis of the physically-based rendering algorithms [27, p. 281], which will be discussed in subsequent

chapters. Here we will briefly go over the related concepts and terminology, along with their definitions.

For our purposes, simulating only the interaction of light with relatively large objects (larger than the light's wavelength) is of concern; hence, radiative transfer is a good enough model to describe these interactions at the level of geometrical optics [27, p. 281]. We will also make the simplifying assumption that electromagnetic radiation consists of a flow of photons and disregard their wave-particle duality; doing this we will not be able to account for phenomena like polarization, interference, and diffraction, which are not necessary for rendering [26, p. 202].

In rendering, we are mainly concerned with four basic radiometric quantities [27, p. 282]L

1. Radiat Flux (ϕ).
2. Radiant Flux Area Density.
   (a) Outgoing, called Radiant Exitance, or Radiosity (B).
   (b) Incident, called Irradiance (E).
3. Radiant Intensity, or Flux Density per Solid Angles (I).
4. Radiance (L).

An more detailed discussion on the more basic radiometric quantities starting from the frequency, wavelength, and energy of a single photon to radiance can be found in [13, p. 13-17], but here we will only discuss the above four quantities; material presented here is based on [26, 13, 27].

Radiant flux of a light source, also known as radiant power, is the total amount of energy Q, passing through space per unit time,

$$\phi = \frac{dQ}{dt} \tag{2.18}$$

Radiant flux is measured in Joules per second ($\frac{J}{s}$), or Watts (W).

Radiant flux area density is defined as the differential Radiant Flux $d\phi$ per differential area dA at a real or imaginary surface in space, and is measured in $\frac{W}{m^2}$,

$$RadiantFluxAreaDensity = \frac{d\phi}{dA} \tag{2.19}$$

Incident and outgoing radiant flux area densities are often distinguished: if the flux is incident on the surface this quantity is called irradiance, denoted by $E(p, \vec{\omega_i})$, and if it is leaving the surface, the quantity is called radiant exitance, or Radiosity, and denoted by $M(p, \vec{\omega_o})$ (or B in some texts), although some sources use irradiance to refer to both incident and outgoing flux. Using the above definition, we can see why the amount of energy received from a light source falls off with the squared distance from the light. We can measure the irradiance of a light source at a distance r by imagining a sphere of radius r centered at the light source; therefore the irradiance reaching the surface of this sphere equals

$$E = \frac{\phi}{4\pi r^2}$$

As we get further from the light source the radius of this imaginary sphere increases and as a result irradiance falls off with the squared distance.

Radiant intensity (I), or simply intensity of a light source is defined as the radiant flux per unit solid angle $\vec{\omega}$,

$$I(\vec{\omega}) = \frac{d\phi}{d\vec{\omega}} \tag{2.20}$$

Radiant Intensity can also be defined as the flux density with respect to the area on an enclosing unit sphere; this means the radiant intensity can also be calculated using irradiance (E),

$$E(r) = \frac{I}{r^2} \Rightarrow I = E(r) \times r^2 \tag{2.21}$$

And finally, radiance (L), is the most important radiometric quantity in rendering [27].

Radiance is the radiant flux per unit solid angle per unit projected area,

$$L(x, \vec{\omega}) = \frac{d\phi}{d\vec{\omega}.dA^{\perp}} \tag{2.22}$$

Where $dA^{\perp}$ is the projected area of dA on a hypothetical surface perpendicular to $\vec{\omega}$. The reason radiance is called the most important radiometric quantity is that it represents an object's color; it can be thought of as the number of photons arriving per unit time at a small area from a given direction. It can also be used to describe the intensity of light at a given point in space in a given direction. Another important characteristic of radiance is that it always remains constant through space for a light source. When dealing with equations involving radiance, it is necessary to distinguish whether the radiance is incident on the surface or outgoing; incoming radiance at point p in direction $\vec{\omega}$ is denoted by $L_i(p, \vec{\omega})$, while the outgoing radiance is denoted by $L_o(p, \vec{\omega})$, such that

$$L_i(p, \vec{\omega}) \neq L_o(p, \vec{\omega}) \tag{2.23}$$

But,

$$L_i(p, -\vec{\omega}) = L_o(p, \vec{\omega}) \tag{2.24}$$

2.7 Surface Reflection and Transmission

2.7.1 Reflection Types

In general surface reflection is split into two broad categories: diffuse reflection and specular reflection, illustrated in Figure 2.1.

Perfectly diffuse surfaces scatter the incident light equally in all directions, also called a Lambertian reflection [13]. The reflection (or outgoing) angle, θ_o, can be any direction chosen uniformly at random from the upper hemisphere centered at the incident point. Although

in practice an ideal diffuse surface does not exist, for the purpose of this thesis we assume a Lambertian reflection when talking about diffuse surfaces unless otherwise noted. Matte paint is an example of a diffuse surface.

Specular surfaces scatter most of the incident light in a particular direction (or set of directions), resulting in a glossy surface. Mirrors are examples of perfectly specular surfaces, reflecting all the incident light with the same angle, meaning $\theta_i = \theta_o$; surfaces which are not perfectly specular, are called glossy specular (e.g. plastic or metallic surfaces), for which the reflection is somewhere in-between perfectly diffuse surfaces and perfectly specular surfaces.

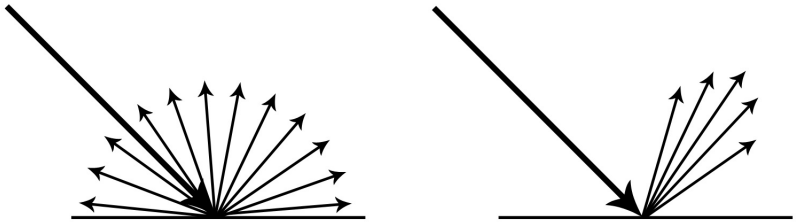

Figure 2.1: Left: diffuse surfaces scatter incident light uniformly in all directions. Right: specular surfaces scatter most of the incident light in particular set of directions.

A more detailed discussion on this subject can be found in [27, Chapter 8], which categorizes the reflection of light into Diffuse, Perfect Specular, Glossy Specular, Retro-Reflective (reflecting light back towards the light source).

2.7.2 Transmission or Internal Reflection

Transmission of light, or internal reflection, is the broad case when the incident light enters a surface, travels inside the surfaces, and exits from the opposite side, e.g. glass or water.

Snell's Law (or Law of Refraction) describes this behavior of light based on the difference between the index of refraction of the medium light is traveling in, and the index of refraction of the medium the light will enter [27],

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t \tag{2.25}$$

where,

- η_i is the refractive index of the medium light is traveling
- η_t is the refractive index of the medium light is entering
- θ_i is the angle between the incident light and surface normal
- θ_t is the angle between the transmitted light and surface normal

The index of refraction describes how much slower light travels in a medium compared to the speed of light in vacuum (c). Internal reflection happens when the light traveling in a medium with a relatively higher index of refraction, reaches a medium with a relatively lower index [26].

2.7.3 Bidirectional Reflectance Distribution Function (BRDF)

Bidirectional Reflectance Distribution Function, or BRDF, describes the behavior of light incident at a surface. A BRDF takes in two parameters: the direction of the incoming light and the direction of the outgoing light, and returns the ratio of the outgoing radiance $Lo(p, \vec{\omega_o}$ to the irradiance $E(p, \vec{\omega_i})$[26, 27],

$$f_{brdf}(\vec{\omega_i}, \vec{\omega_o}) = \frac{dLo(p, \vec{\omega_o}}{dE(p, \vec{\omega_i})} \tag{2.26}$$

BRDFs have to important properties [27]:

1. Helmholtz Reciprocity, meaning have $f_{brdf}(\vec{\omega_i}, \vec{\omega_o}) = f_{brdf}(\vec{\omega_o}, \vec{\omega_i})$.
2. Energy conservation, meaning for any surface that is not a light source itself, the outgoing radiance is always less than the incident radiance.

There are two other widely used models for describing the light's behavior at a surface:

1. Bidirectional Transmittance Distribution Function, or BTDF, describes the behavior of light at surfaces that transmit the incoming light, e.g. glass or water.

2. Bidirectional Scattering Distribution Function, or BSDF, which is a generalization of BRDFs and BSDFs, used to describe surfaces exhibiting both behaviors (reflection and transmission)[27].

2.8 Light Transport Equation

The rendering equation, also called the light transport equation (LTE), forms the mathematical basis for all global illumination algorithms [13], which all the Monte Carlo image synthesis algorithms try to solve. The rendering equation is an integral equation, introduced by Kajyia in [20], which generalizes a variety of known rendering algorithms and models a wide variety of optical phenomena by performing a geometrical optics approximation. It can be used to compute the outgoing radiance at any point on any surface in a scene[13]; it states that the outgoing radiance or light transport intensity from a surface point x to another surface point x' is the sum of the emitted light and the total light intensity which is scattered toward x from all other surface points [20].

Although different forms and notations of Kajyia's equation is used throughout the literature, I find the form and notation used in [31] to be more descriptive:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{S^2} f_r(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| d\omega_i, \tag{2.27}$$

where $L_o(p, \omega_o)$ is the outgoing radiance from point p in direction ω_o, $L_e(p, \omega_o)$ is the emitted radiance from p in direction ω_o (in case the surface is a light source), the domain S^2 is the hemisphere centered at point p, $L_i(p, \omega_i)$ is the incident radiance at p from some direction ω_i in the domain scaled by f_r (BRDF at point p) and $\cos \theta_i$ (θ_i being the angle between surface normal at p and ω_i). Therefore what (2.27) describes is simply as follows: the outgoing radiance at some p in some direction ω_i is the sum of the emitted light in that direction from that point, and the aggregate of all the radiance incident on p, taken into account the surface's BRDF and θ_i.

The rendering equation in its original form does not attempt to model all interesting optical phenomena; it only model time averaged transport intensity and does not take phase and diffraction into account; it also assumes that the media between surfaces (usually air) is of homogenous refractive index and does not itself participate in the scattering of light [20].

Evaluating LTE is the main reason we are interested in Monte Carlo integration method, and the reason MC rendering methods dominate the field of physically-based image synthesis.

2.9 Sampling Methods

Having discussed the Monte Carlo integration method and the light transport equation, now we need to discuss the methods by which we take the samples to complete this part of the discussion. One way to get a good estimate for LTE (2.27) using the Monte Carlo integration method (2.17) is to make N as big as possible (i.e. use as many samples as possible), but in practice this is not efficient, and arguably not practical, and therefore not desirable. As it was mentioned earlier in 2.3, another way to improve the accuracy of the estimate is to take samples such that $\frac{g(x_i)}{p(x_i)}$ in 2.17 has a low variance [19]; different sampling methods have been introduced to achieve this goal, and we are going to briefly go over a number of them in this section, which are often used in the literature. The contents of this section are based on [27, 7, 19, 22, 21, 29].

2.9.1 Uniform Sampling

The simplest method of sampling is to construct samples using independent and uniformly distributed random numbers. The most obvious disadvantage of this method is that it does not guarantee to sample a function according to the function's shape, e.g. if we are sampling a Gaussian function, this method will result in over-sampling and under-sampling of the function. A more specific example would be the case of sampling the image plane, a uniformly distributed random 2D point on the surface of a particular pixel is chosen as a

sample. The problem arises when we try to take more samples using this method: how can this method guarantee that samples are not located close to each other?

Using this method, we may end up taking a large amount of samples in one subregion of the pixel of interest, while ignoring the rest of that pixel which may (or may not) be of more importance; for instance imagine a case where shooting a ray into the scene through point p_j on some pixel P_i, hits a surface that is not visible from any light sources, if we continue to take samples that happen to be located close to p_j then the final color of P_i will end up darker than it really is. To solve this problem, one obvious way is to take a large number of samples to make sure we are also taking into account the samples with more contribution.

Same problem exists when sampling an area light source, or a hemisphere centered at an intersection point in order to calculate indirect illumination, etc. It must be clear that the rate of convergence for this method is very low. To address this issue, we use other sampling methods which will be discussed in the following subsections.

2.9.2 Stratified Sampling

Stratified sampling is one of the simplest ways to address the shortcomings of the uniform sampling, by dividing the domain of interest into subregions, or strata, and then use another sampling method (e.g. uniform sampling) on each of these strata to "forcefully" scatter samples over the domain in order to make sure they are not concentrated in one area.

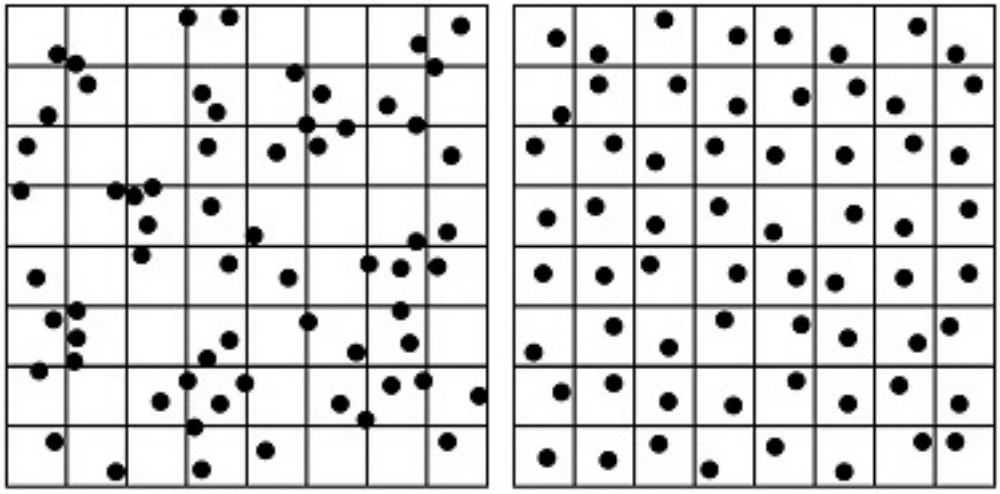

Figure 2.2: Left: an $8x8$ plane sampled using uniform sampling; note that some grid cells have no samples, while some have multiple samples. Right: same plane sampled using stratified sampling; a sample is taken inside each cell.

To continue with the pixel example mentioned earlier, a pixel can be divided into multiple subpixels, each subpixel being treated as a pixel itself; then applying another sampling method over the subpixels we can avoid the hypothetical problem described above. Same technique can be applied to sampling an area light source, or a hemisphere: divide the area of interest into subregions, use other methods to sample each subregions.

2.9.3 Importance Sampling

Importance sampling is a method that attempts to reduce the variance of the estimate by choosing $p(x_i)$ in (2.17) intelligently; this means that we attempt to choose a density $p(x)$ that has a similar shape to $g(x)$, is simple, and is efficient to evaluate [19, 22].

Using this method, we can try to avoid paying the extra cost of taking into account samples which make relatively low contributions to the final estimate; instead we increase the chance and weight of the samples with high contributions.

For instance, consider sampling directions over the hemisphere centered at some intersection point p; here we can use the Cosine of the angle θ between the surface normal $\vec{N}$ at p and the sample (i.e. direction) as our density function according to which we choose

our samples: as the angle increases, $\cos\theta$ decreases, hence the contribution of light in that direction decreases. This will result in taking more samples where $\cos\theta$ is higher hence the contribution of light is higher [27, 19].

One disadvantage of importance sampling is that a poorly chosen density can result in higher variance, in which case stratified sampling (or even uniform sampling) would end up being much more effective than importance sampling.

2.9.4 Multiple Importance Sampling

Looking at the light transport equation (2.27) we notice that the integrand is a product of three different functions:

$$L_r = f_r(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos\theta_i|, \tag{2.28}$$

which begs the question whether or not using a single importance density function (e.g. a Cosine importance function) would result in a good enough estimate? In the best-case scenario, we may manage to choose a density that is close to (2.28), but in practice this can rarely happen [22]. However, the form of the integrand can potentially provide some opportunities to use more than one importance functions together.

As an example, imagine the situation where the point of interest p lies on a near-perfect specular surface; if we based the importance density on f_{BRDF} at p, we are effectively ignoring all the incoming radiance that is outside the range of f_{BRDF}.

Multiple importance sampling (MIS), introduced by Veach and Guibas in [31, 32], addresses the above issue: take samples using more than one importance density, and weight samples according to their respective densities such that the final estimate has the lowest variance. In MIS, a variety of sampling techniques (even for special cases) can be used, wherever one (or more) particular method is not desired, the fact that we are weighting the samples will take care of the undesired effects[27].

We can rewrite the Monte Carlo integration estimator (2.17) with MIS as follows[27, 32]:

$$\int_{x\in S} f(x)g(x)dx \approx \frac{1}{N_f}\sum_{i=1}^{N_f}\frac{f(x_i)g(x_i)\omega_f(x_i)}{p_f(x_i)} + \frac{1}{N_g}\sum_{j=1}^{N_g}\frac{f(x_j)g(x_j)\omega_g(x_j)}{p_g(x_j)} \tag{2.29}$$

where N_f is the number of samples taken from density p_f weighted by ω_f, and N_g is the number of samples taken from density p_g weighted by ω_g.

In [32], Veach and Guibas describe three methods to compute ω:

1. Balance heuristic,

$$\omega_i = \frac{N_i p_i}{\sum_j N_j p_j} \tag{2.30}$$

2. Cutoff heuristic, discard samples with low weights,

$$\omega_i = \begin{cases} 0, & \text{if } p_i < \alpha\ p_{max} \\ \sum_j \left\{p_j \mid p_j \geq \alpha p_{max}\right\}, & \text{otherwise} \end{cases} \tag{2.31}$$

   Where $p_{max} = max_j p_j$, and α is a constant deciding the lower-bound for the cutoff.

3. Power heuristic, raise all weights to a power β, and then normalize,

$$\omega_i = \frac{N_i p_i^\beta}{\sum_j N_j p_j^\beta} \tag{2.32}$$

For detailed proof and an in-depth discussion please refer to [31, 32, 27, 22].

3. Monte Carlo Ray Tracing Methods

3.1 Precursors to MCRT

This section contains a brief overview of the methods introduced in 1970s and 1980s which are precursors to MCRT methods, although none of the algorithms mentioned in this section are directly implemented as part of my work. Having a general understanding of these methods paves the way for understanding more advanced approaches such as Path Tracing and Metropolis Light Transport.

Taking global illumination information into account to accurately render a scene was first discussed in Turner Whitted's seminal paper titled An Improved Illumination Model for Shaded Display published in 1979. It is worth noting that Whitted's method, commonly known as Recursive Ray Tracing, is partially based on Arthur Appel's 1968 paper titled Some Techniques for Shading Machine Renderings of Solids which introduced a much simpler method commonly known as Ray Casting.

Whitted's paper [36] has two main contributions,

- A new shading model.
- A new visible surface detection algorithm based on tracing rays from camera into the scene.

Whitted's proposed a new shading model is summarized in this equation:

$$I = I_a + k_d \sum_{j=1}^{j=I} (\vec{N}.\vec{L_J}) + k_s S + k_t T \tag{3.1}$$

Where I is the reflected intensity, I_a is the reflection due to ambient light, k_d denotes the diffuse reflection constant, $\vec{N}$ is the unit surface normal, $\vec{L_J}$ is the vector in the direction of

Jth light source, S is the intensity of specular light, k_t is the transmission coefficient, and T is the intensity of transmitted light.

To make the surface less glossy, k_s should be smaller and k_d should be larger. Note that Whitted's illumination model improves the methods introduced by Phong and Blinn, using a recursive algorithm; (3.1) in itself does not take global illumination into account rather it approximates the reflection from a single surface. In a real scene with moderate complexity light will reflect off of several surfaces, causing inter-reflection, before reaching the viewer, therefore the above model needs to be evaluated for each surface that takes part in these inter-reflections.

Whitted proposed building a tree to represent the inter-reflection of rays from a light source to the viewer. Creating this tree requires calculating the point of intersection of each component ray with the surfaces in the scene. This is done by recursively calling the visible surface detection algorithm which was proposed in the same paper.

The visible surface detection algorithm is based on Arthur Appel's aforementioned 1968 paper [2]: trace rays through image plane into the scene and find the nearest ray-object intersection point. Whitted adds the idea of calling the method recursively for each intersection found, and building the tree of inter-reflections using this method. Whitted proposes the use of bounding volumes (e.g. axis-aligned bounding volumes) to reduce unnecessary ray-object intersection tests which is itself based on a previous work [5] published in 1976.

Once the tree is formed, the shading model is applied to each intersection point (i.e. each tree node), taking the attenuation of light into account based on the intersection point's distance from the light source.

Ray tracing is a precise algorithm to render a scene. Although this is essential for generating realistic images, it has some limitations to simulate the behavior of light in the real world. Rendered scenes with ray tracing have precise and sharp shadows, sharp reflections, sharp refractions, cannot handle motion blur, and have some difficulties showing depth of field. Simulating these "fuzzy phenomena" was addressed by Distributed Ray Tracing [8].

In distributed ray tracing, the rays are distributed in a time frame so that every sampling happens in a different time frame. Using this approach, it is easy to generate the following fuzzy effects without any extra sampling efforts:

- Blurred reflection by sampling the reflected ray according to the specular distribution function produces gloss.
- Blurred transparency by sampling the transmitted ray produces translucency.
- Penumbras by sampling the solid angle of the light sources.
- Depth of field by sampling the camera lens area.
- Motion blur by sampling in time.

With this method, an analytic function is used to calculate each pixel's intensity; this function consists of integrations over pixel region, time, lens area, and so son. Although these integrations could be difficult to compute, it is possible to solve them using point sampling.

Another method was introduced by John Amanatides, called Ray Tracing with Cones (or more commonly, Cone Tracing) to address the problem of sharp shadows in ray-traced images [1]. The proposed method is to redefine light rays as cones. Using this approach, there is no need for super-sampling to generate soft shadows. Instead, at the intersection point of ray and surface, the portion of the ray (i.e. cone) which is in intersection area can be calculated instead, more intersection area means softer shadows. Although there are workarounds for the conventional recursive ray tracing to render soft shadows (anti-aliasing), this new method reduce the computations significantly and is faster. Besides, considering rays as cones, a renderer can control level of details in the scene by manipulating the size of the cone for each ray.

Another refinement for Whitted's method was introduced in 1986 by James Arvo [3]; ray tracing cannot simulation the diffuse reflection of indirect lights properly. To solve this problem, Arvo proposed a new method in which a scene is rendered in several phases. In

the pre-processing phase, an illumination map is created to help the final phase to better simulate indirect diffuse reflection. James Arvo called this method Backward Ray Tracing, in contrast to Whitted's ray tracing, which Arvo labeled Forward Ray Tracing.

In Arvo's backward ray tracing, the rays are traced from the light source to the objects. In the pre-processing step, a large number of rays are emitted from the light sources and the reflections and refractions are then calculated recursively. At each reflection and refraction step, the energy of the ray is reduced, to mimic the natural behavior of light. The result of this phase is an illumination to be used by the next phase: forward ray tracing; therefore the ray tracing algorithm runs at least twice in Arvo's method, one time from the point of view of each light source in the scene, and the last phase from the camera.

Radiosity is another precursor to MCRT methods, introduced in 1984 by Goral et al. [9]. They proposed a method to solely model the interaction of light between diffuse surfaces. This method is based on thermal engineering and heat transfer, which was a completely different approach compared to other global illumination methods at the time. The key point of this approach is to simulate what exactly happens in the real environment by considering the total energy of the light in the scene. This method assumes that the surfaces in the scene are ideal diffuse reflectors, although in reality such a surface does not exist. Radiosity tries to model the object-to-object reflections between surfaces in a physically-correct manner, therefore it is considerably slower than the traditional ray tracing. One of the most important characteristics of this method is that the rendering process is independent of the viewers position, meaning it can be used as a precomputed lighting effect to use in a scene where the observer can move without having to re-render the scene.

3.2 Path Tracing and The Rendering Equation

In 1986 Kajyia introduced the Rendering Equation which forms the mathematical basis for almost all of the global illumination algorithms proposed since then. We briefly discussed

the rendering equation in Section 2.8, here we are going to discuss the equation as it was first introduced by Kajyia in [20].

The rendering equation is an integral equation generalizing a variety of known rendering algorithms, and models a wide variety of optical phenomena by performing geometrical optics approximations. It can be used to compute the outgoing radiance at any point on any surface in a scene; it states that the outgoing radiance or light transport intensity from a surface point x to another surface point x' is the sum of the emitted light and the total light intensity scattered toward x from all other surface points:

$$I(x, x') = g(x, x') \times \left(\epsilon(x, x') + \int_S p(x, x', x'').I(x', x'')dx'' \right) \tag{3.2}$$

In which:

- $I(x, x')$ measures the energy of radiation passing from point x to point x', called un-occluded two point transport intensity from x' to x. It's unit is $\frac{Joules}{m^4 sec}$.

- $g(x, x')$ is a geometry term that encodes the occlusion of surface points by other surface points; if it is 0 then surface points x and x' are not mutually visible, otherwise it will be equal to $\frac{1}{r^2}$ where r is the distance between two points (attenuation of light, refer to 2.6).

- $\epsilon(x, x')$ is the emittance term which measures the energy emitted by a surface at point x' reaching a point x. It is called the un-occluded two point transport emittance from x' to x. It gives the energy per unit time per unit area of source and per unit area of target; its unit is $\frac{Joules}{m^2 sec}$.

- $p(x, x', x'')$ is called the scattering term or more precisely un-occluded three point transport reflectance from x'' to x through x'; it is the intensity of energy scattered by a surface element at x' originating from a surface element at x'' and terminating at a surface element at x.

- S is the domain of integration and is the union of all surface including a global background surface S_0 which is a large enough hemisphere to act as an enclosure for the entire scene.

The above quantities (except for S) are called un-occluded multipoint transport quantities [20].

The rendering equation in its original form does not attempt to model all interesting optical phenomena; it only models time averaged transport intensity and does not take phase and diffraction into account. It also assumes that the media between surfaces (usually air) is of homogenous refractive index and does not itself participate in the scattering of light[20].

Kajiya in the same paper introduces a new global illumination method, which today is commonly referred to as Path Tracing, which makes [20] the first to propose a general-purpose Monte Carlo approach to image synthesis.

Path Tracing is a straightforward extension to ray tracing that makes it possible to compute lighting effects that requires evaluating integration problems such as area lights and indirect light reflected by diffuse surfaces. Mathematically, path tracing is a continuous Markov Chain random walk technique for solving the rendering equation, in which the unknown function used in Monte Carlo integration to approximate the integral part of the rendering equation is sampled by tracing rays stochastically along all possible light paths. By averaging a large number of sample rays for a pixel we get an estimate of the integral over all light paths through that particular pixel [13].

Like ray tracing, path tracing starts by shooting a number of rays from the view point into the scene and continues by tracing each ray until it can reach a light source; however, there are a number of differences between Whitted's ray tracing and path tracing [13]

1. The illumination model; for path tracing it is described by the rendering equation whereas ray tracing's illumination model is Whitted's improved model based on Phong shading.

2. The paths these two algorithms choose in order to shade pixels; for Whitted's ray tracing the form of each path can be described as $E[S*](D|G)L$ and for path tracing it can be described as $E[(D|G|S) + (D|G)]L$

Path tracing algorithm works well for many scenes, but it can exhibit high variance in the presence of particular tricky lighting conditions; for instance consider a scene where a light source only illuminates a small area on the ceiling and the rest of the room is illuminated by the indirect lighting bouncing from the illuminated area on the ceiling. Using path tracing we will almost never happen to sample a path vertex in the illuminated region on the ceiling before we trace a shadow ray to the light. Most of the paths will have no contribution, while a few of them will have a large contribution. The resulting image will have high variance [27].

Most important disadvantages of the Path Tracing algorithm are the followings:

- Rendering is time consuming (very slow convergence)

- Variance in the estimation will show up as noise in the final image; this can be eliminated by using large number of samples per pixel; in cases of complex scenes a noise-free final image can be rendered by using 1,000 to 10,000 paths/pixel, and even more than this amount for scenes with higher complexities.

- Although it can trace a wide variety of paths with different patterns, it ignores the paths of the form E(D—G)S*L; it means that paths containing multiple specular bounces from the light source (e.g. in caustics) are ignored.

A number of methods have been proposed such as reusing paths [4] and use of noise reduction filters [18] to deal with Path Tracing's efficiency and noise issues.

3.3 Bidirectional Path Tracing

Bidirectional Path Tracing (BPT) is an extension to the path tracing algorithm, which was introduced first by Lafortune and Willems [23], and advanced further in [24, 31, 32].

BPT is a Monte Carlo rendering algorithm which combines the advantages of algorithms based on the camera position (such as ray-tracing and path tracing) and the algorithms based on the position of the light sources in the scene (such as radiosity) [23], in order to address the problem of slow rate of convergence in path tracing.

To better understand the advantages of bidirectional path tracing, imagine an in-door scene illustrated by Figure 3.1, where a room is illuminated only by indirect lighting (e.g. one or more light sources located outside the room, and a window or door frame is the only way any light ray can find its way inside the dark room). Now consider rendering this scene using path tracing (or distributed ray tracing), we start from the camera, shoot rays inside the dark room, and try to connect the intersection points directly to a light source. There is a high probability that we won't be able to find any visible light sources from the intersection point. Same problem persists when we continue with computing the diffuse reflection component, no matter how many bounces taken into account, there is a very low probability of ever finding a light source located outside this room. This will result in high variance and the final image will end up extremely noisy.

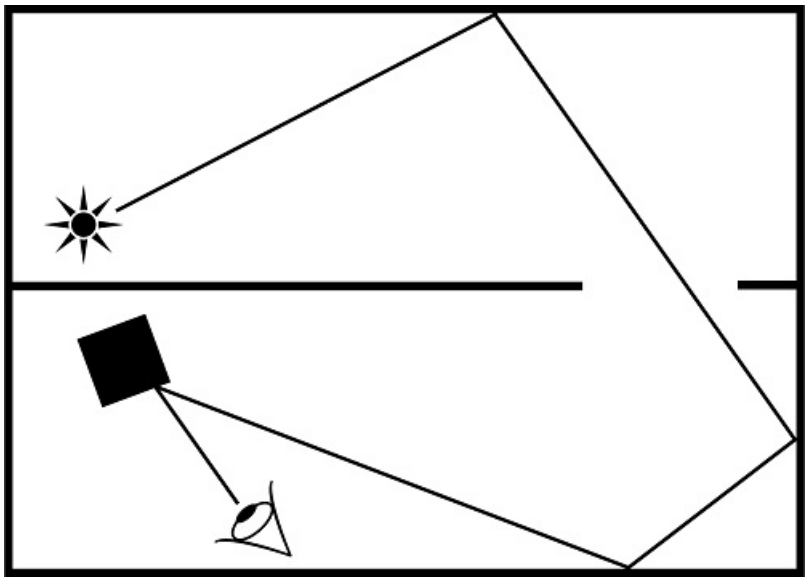

Figure 3.1: Example of a scene lit almost completely by indirect lighting.

It must be obvious now that taking the position of the light sources into account is important to get the best estimate possible and lower the variance. This is exactly what bidirectional path tracing tries to add to the original path tracing method.

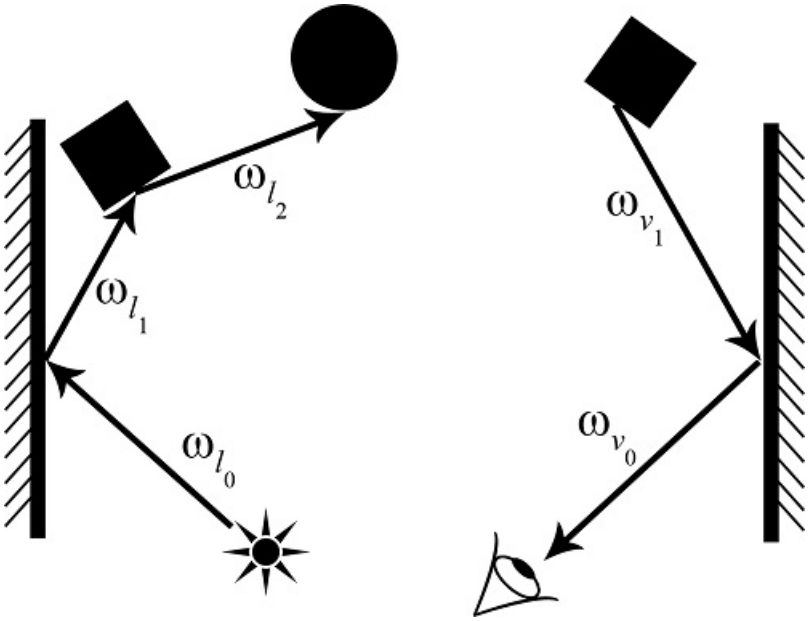

Figure 3.2: Constructing paths in BPT.

The way BPT works is to shoot rays both from the camera position and a selected light source at the same time, bounce the light from each of these starting points independently

for some number of iterations and then connect the ending vertexes using a visibility ray. The algorithm then proceeds to connect vertexes on one paths to all the vertex on the other path [23, 27], forming the graph shown in Figure 3.2.

Here the path starting from camera is denoted by $v_0, v_1, ..., v_{N_i}$, where each v_{i+1} is a point visible from point v_i along direction $\omega_{v_{i+1}}$; note that the first point v_0 is not located on the camera itself but on the first intersection of ray shot from the camera. The path starting from the light source is denoted by $l_0, l_1, ..., l_{N_j}$, where each l_{j+1} is a point visible from point l_j along direction $\omega_{l_{j+1}}$; note that the first point l_0 is located on the light source itself, meaning we need to sample the area light source's surface to begin the path.

Paths in Figure 3.2, are formed by performing random walks according to appropriately chosen probability distribution functions at each of the vertexes. To start building these paths, we first must find the initial points v_0 and l_0.

As described by Lafortune and Willems in [23], to sample v_0 and ω_{v_0}, we can use the following *pdf*:

$$pdf(v, \omega_v) = \frac{g(v, \omega_v)|Cos\theta_v|}{G} \tag{3.3}$$

where θ_v is the angle between ω_v the normal at v, $g(v, \omega_v)$ is a function that is 1 for all the contributing pairs of points and directions (e.g. not obstructed, not in the shadow, etc.) and is 0 otherwise, and G is the normalization factor of the *pdf*.

We can sample l_0 and ω_{l_0} using a similar *pdf*:

$$pdf(l, \omega_l) = \frac{L_e(l, \omega_l)|Cos\theta_l|}{L} \tag{3.4}$$

where θ_l is the angle between ω_l the normal at l, L_e is the self-emitted radiance at point l in direction ω_l, and L is the normalization factor of the *pdf*. This *pdf* is derived based on importance sampling method ensuring both amount of emitted light and direction are sampled where the light is brighter.

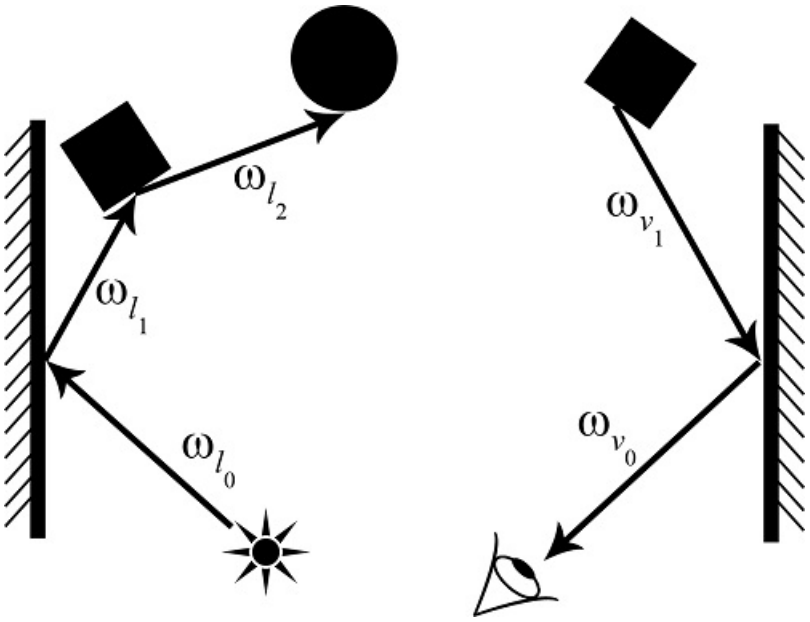

Figure 3.3: Having formed the two paths from a light source and the camera, BPT proceeds by connecting all the vertexes on these paths using visibility rays.

Having v_0 and l_0, we can start the random walk by sampling directions ω_{v_1} and ω_{l_0} according to the BRDFs at points v_0 and l_0, then find the intersection point, and repeat the process for as many iterations as needed (light source path and camera path need not to be of the same length):

$$pdf(\omega_v) = f_{brdf}(v_i, \omega_v, \omega_{v_i})|Cos\theta_{v_i}| \tag{3.5}$$

$$pdf(\omega_l) = f_{brdf}(l_{i+1}, \omega_{l_i}, \omega_l)|Cos\theta_{v_{i+1}}| \tag{3.6}$$

Having formed both paths, next step is to connect all the points using visibility rays as illustrated in Figure 3.3, then add the weighted partial contributions of each subpath to calculate the final estimate,

$$F = \sum_{i=0}^{N_l} \sum_{j=0}^{N_v} \omega_{i,j} C_{i,j} \tag{3.7}$$

where $C_{i,j}$ denotes the partial estimate for i random walks on the light source path and j random walks on the camera path, and $\omega_{i,j}$ is the weight associated with this partial estimate. Without going into details of computing the partial contributions, it is worth noting that

three cases are possible based on number of random walks performed:

- Direct contribution of light source to v_0 when $i = 0$ and $j = 0$, meaning v_0 is directly illuminated by the light source.
- Special case, where $i = 0$ and $j > 0$, making bidirectional path tracing equivalent to pure path tracing.
- General case, where both $i > 0$ and $j > 0$.

For detailed discussions and proofs on computing the weights and partial contributions refer to [23, 24, 31, 32].

Although bi-directional path tracing results in less noisy images compared to the path tracing algorithm, the resulting image still end up being noisy if enough samples are not taken for each pixel.

3.4 Photon Mapping

Photon mapping is one of the classical Monte Carlo rendering algorithms which was introduced by Henrik Wann Jensen in [15] based on his earlier published work in [16, 14, 17]. In this section we will discuss the classical photon mapping algorithm as it was introduced by Jensen, and will separately discuss an improvement on this algorithm at the end of this chapter (Section 3.6), namely Progressive Photon Mapping [11].

Photon mapping is a two-step biased Monte Carlo rendering algorithm which takes advantage of the concept of photon maps to optimize the sampling process in Monte Carlo ray tracing algorithms, deal with the noise problem in unbiased MC methods, speed up the process, and to deal with rendering caustics which can be extremely difficult or even in certain scenarios almost impossible to achieve using unbiased Monte Carlo ray tracing algorithms, this is true even for methods that were introduced years after photon mapping such as Metropolis light transport and energy redistribution path tracing. However, note that

photon mapping sacrifices the unbiased property of other Monte Carlo rendering algorithms such as path tracing and Metropolis light transport to achieve this goal, therefore the results are not guaranteed to be correct but may look visually plausible.

Before going into details about the algorithm itself, it is necessary to understand why rendering caustics using path tracing (or similar unbiased Monte Carlo rendering algorithms) is extremely inefficient and may even in some cases be practically impossible. This particular problem sometimes is referred to as SDS path problem [11]. An SDS path in Heckbert's notation is a light transport path from a specular surface to a diffuse surface and then again to a specular surface and from there to the camera. For instance, any specular surface seen by the camera in a scene where the light source is a light bulb (light source inside a glass bulb) is an example of an SDS path.

Imagine the scenario illustrated in Figure 3.4; a light ray emitted from the light source hits a glass surface, is refracted inside the glass object, again hits another side of that glass objects and exits the object hitting a diffuse surface, bounces towards another highly glossy surface (e.g. a mirror) before bouncing towards the camera.

Now let us try and trace this path from the camera: first we sample the pixel area, shoot a primary ray inside the scene through that sample. Let's assume this ray happens to hit the mirror at point p_1, then we are bound to perform a mirror reflection, which will hit the point p_2 on the diffuse surface. Now here is where the problem arises: we have a reflection at a diffuse surface, therefore we need to randomly choose a direction in the upper hemisphere centered at p_2; the probability of sampling the exact direction to form the rest of the path shown in Figure 3.4 over the continuous domain of all the possible directions in the hemisphere is exactly zero if the light source if infinitely far from the surface, and it is very small for area light sources (the probability gets worse as the light source gets smaller).

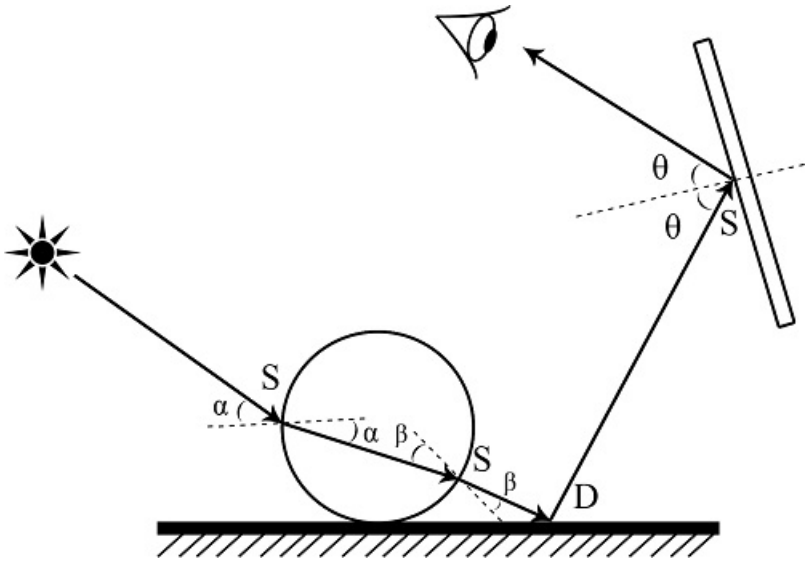

Figure 3.4: This figure illustrates why rendering caustics can be very difficult and extremely inefficient.

Intuitively, we can see that the biggest issue with rendering the above scenario, is that after the diffuse reflection at p_2, we cannot guarantee to randomly sample the correct direction which in turn makes yet another mirror reflection towards the light source. One of the main strong points of photon-mapping-based algorithms is the ability to deal with these types of situations by introducing bias into the estimate.

Photon maps were introduced in [16] by Jensen and Christensen as a new way to store irradiance information of surfaces, and as an improvement on James Arvo's illumination maps [3]. A photon map is a space partitioning data structure named kd-tree which is used to store photons and some information about each photon such as the intersection point, normal, and energy.

As it was mentioned in the beginning of this section, photon mapping is a two-step rendering process:

1. Constructing two photon maps

2. Rendering the scene using a Monte Carlo ray tracing algorithm

In the first pass, a large number of photons (packets of energy) are emitted towards the

scene from the light sources. Tracing these photons using a ray-tracing-based method (e.g. path tracing), we proceed to store the photons in the photon maps as they hit surfaces. Russian Roulette is used to decide at each intersection whether or not the photon gets absorbed by the surface or is reflected.

Two photon maps are stored in the first pass, one high resolution photon map for caustics rendering named Caustics Photon Map, and one low resolution photon map for other cases named Global Photon Map.

Caustics photon map is constructed by emitting a high density of photons toward specular surfaces; a photon is stored in the caustics map if after hitting the specular surface, it hits a diffuse surface (e.g. the problem we discussed earlier in this section). We will use this map exclusively to render caustics in the second pass.

Global photon map is constructed by emitting photons toward all the objects in the scene. Two types of photons are stored in this case: illumination photons and shadow photons.

The concept of shadow photons was introduced by Henrik Wann Jensen in [17] as a way to capture shadow information in a scene inside a photon map and eliminate the need for casting shadow rays (e.g. in most ray-tracing-based algorithm). The difference between a illumination photon and a shadow photon is that former contains a positive amount of energy but the latter contains a negative amount of energy. A shadow photon is stored by emitting photons from light sources for one single pass then storing an illumination photon at each hit point, but shadow photons are stored at all the hit points formed by the consequent reflections of these photons.

Second pass of this algorithm consists of rendering the scene by a Monte Carlo ray tracing algorithm (e.g. path tracing), using the two photon maps constructed in the first pass to speed up the process of evaluating the light transport equation.

In [15], the integral part of the rendering equation (introduced in Section 2.8) which computes the total incoming radiance at the hit point p over the upper hemisphere centered

at p, is split into a sum of four integrals as follows:

$$\begin{aligned} L_r = & \int_{S^2} f_r L_{i,l} |\cos\theta_i| d\omega_i + \\ & \int_{S^2} f_{r,s}(L_{i,c} + L_{i,d}) |\cos\theta_i| d\omega_i + \\ & \int_{S^2} f_{r,d} L_{i,c} |\cos\theta_i| d\omega_i + \\ & \int_{S^2} f_{r,d} L_{i,d} |\cos\theta_i| d\omega_i \end{aligned} \tag{3.8}$$

where $L_{i,l}$ denotes incoming radiance contribution directly coming from the light sources, $L_{i,c}$ denotes the specular reflection contributions directly coming from the light sources (caustics), $L_{i,d}$ denotes the indirect illumination (diffuse bounces), $f_{r,s}$ denotes the specular part of of the BRDF, and $f_{r,d}$ denotes the diffuse part. Note that

$$\begin{aligned} L_i &= L_{i,l} + L_{i,c} + L_{i,d} \\ f_r &= f_{r,s} + f_{r,d} \end{aligned} \tag{3.9}$$

The directions stored in the photon maps in conjunction with surface BRDFs are enough to estimate the above integrals (except for the specular radiance case for which a standard Monte Carlo ray tracing algorithm suffices) using nearest neighbor density estimation.

Imagine a point x on a surface with f_r as the BRDF at that point. We form a sphere centered at x, and expand its radius r until exactly N photons in the photon map are contained inside this sphere, then we can use them to estimate the rendering equation as follows:

$$L_r \approx \sum_{p=1}^{N} f_r \frac{\Delta\phi_p(x, \omega_{i,p})}{\Delta A} \tag{3.10}$$

where $\phi_p(x, \omega_{i,p})$ denotes the energy of the photon incident on point x from direction $\omega_{i,p}$, and ΔA is an estimate for the area of the sphere that contains these N photons. Assuming the area surrounding x is completely flat, the photons are located on a circular surface, cross section of the sphere with radius r centered at x, therefore $\Delta A = \pi r^2$.

For a more detailed discussion on this method refer to the four original papers [15, 16, 14, 17] as well as Henrik Wann Jensen's book on the same topic [13].

Photon mapping's biggest weakness is arguably the need for a high resolution/high density photon mapping for rendering caustics. As the caustics effects in a scene gets more complex, more and more memory is needed to get an acceptable result, which is also a problem when it comes to performance since emitting more photons mean more time needed for the rendering process. Another related problem is that the nearest neighbor density estimation performed by photon mapping tends to make the image blurry unless a large number of photons are used.

3.5 Metropolis Light Transport

Metropolis Sampling uses random mutations to produce a set of samples with a desired density. It was introduced in 1953 by Metropolis et al. [25] for handling difficult sampling problems in computational physics and is the basis for the Metropolis Light Transport algorithm [33].

This technique has the ability to generate a set of samples from any non-negative function $f : S \to R$ that are distributed proportionally to f's value; the algorithm can work without any knowledge about the properties of f, it only needs to be able to evaluate it[19].

The Metropolis sampling algorithm generates a set of samples x_i from a function $f : \Omega \to R$ which is defined over an arbitrary-dimensional state space Ω (usually $\Omega = R^n$) and returns a value in real numbers.

Suppose that the first sample point $x_0 \in \Omega$ is selected then Metropolis sampling can generate each subsequent sample x_i using a random mutation over its previous sample x_{i-1} to compute a proposed sample x'; we denote this by $T(x \to x')$. The proposed sample may be accepted or rejected according to an acceptance probability $a(x \to x')$; this acceptance probability should be defined in a way that can ensure the distribution of samples is proportional

to f [27].

The acceptance probability is best obtained by

$$a(x \to x') = min(1, \frac{f(x')}{f(x)}) \tag{3.11}$$

The pseudocode for the basic Metropolis sampling algorithm is

```
x := x_0
for i := 1 to n
    x' := mutate(x)
    a  := accept(x, x')
    if ( random() < a )
        x := x'
    store(x)
```

As it was mentioned earlier, the goal of all global illumination algorithms is to solve the rendering equation for every visible point in the scene; the integral part of the rendering equation cannot be solved by analytical integration methods mainly because the integrand is not always known for every surface in every scene and even if it is known, it is most likely that it cannot be solved analytically; therefore Monte Carlo integration methods are used to approximate this integral equation. For doing so, we must make use of suitable sampling methods to make this approximation more accurate (decrease the variance). In Metropolis Light Transport algorithm (MLT), the sampling process is done using Metropolis Sampling Method which was briefly introduced above.

Metropolis Light Transport was first introduced by Eric Veach and Leonidas J. Guibas in 1997 in a paper[33] of the same title; they recognized that Metropolis Sampling algorithm could be applied to the image synthesis problem after it was appropriately reformulated, and they used it to develop a general and unbiased Monte Carlo algorithm which was named Metropolis Light Transport [13, 33].

To render an image using MLT, the paths are sampled starting from the light sources to the lens (view-point). Each path is $\bar{x}$ is a sequence $x_0 x_1 \cdots x_k$ of points on the scene surfaces, where $k \geq 1$; this sequence of light transport paths are generated by randomly mutating a single current path using a suitable mutation strategy. Each mutation is accepted or rejected with a carefully chosen probability, to ensure that paths are sampled according to the contribution they make to the ideal image. Then the image is estimated by sampling many paths and recording their locations on the image plane [33].

A function f is defined together with a measure μ such that $\int_D f(\bar{x}) d\mu(\bar{x})$ represents the flux that flows from the light sources to the image plane along a set of paths D. The function f is called the image contribution function and it is proportional to the contribution made to the image by light flowing along the path $\bar{x}$ [33].

In MLT another form of the rendering equation is used which essentially the same as Kajiya's original rendering equation (3.2) [33]:

$$L(x' \to x'') = L_e(x' \to x'') + \int_S L(x \to x') f_s(x \to x' \to x'') G(x \leftrightarrow x') dA(x) \tag{3.12}$$

Where,

- S is the union of all scene surfaces
- A is the area measure on S
- $L_e(x' \to x'')$ is the emitted radiance leaving x' in the direction of x''
- $L(x \to x')$ is the equilibrium radiance function
- f_s is the BSDF of the surface point
- Notation $x \to x'$ symbolizes the direction of light flow between two points of S
- Notation $x \leftrightarrow x'$ denotes symmetry in the argument pair
- G represents the throughput of a differential beam between $dA(x)$ and $dA(x')$.

G can be obtained using

$$G(x \leftrightarrow x') = V(x \leftrightarrow x') \times \frac{|\cos\theta_o . \cos\theta'_i|}{||x - x'||^2} \tag{3.13}$$

where θ_o and θ'_i represent the angles between the segment $x \leftrightarrow x'$ and the surface normals at x and x' respectively. $V(x \leftrightarrow x')$ is the visibility test and equals to 1 if two points x and x' are mutually visible and is otherwise zero.

The overall strategy of MLT is to sample paths with a probability proportional to f, and record the distribution of paths over the image plane. As it was mentioned earlier, these paths are a sequence of subpaths each of which is generated by a proposed mutation to its previous subpath which may or may not be accepted according to an acceptance probability.

MLT estimates a finite number of measurements of the equilibrium radiance L (3.12) equal to the number of total pixels in the final image; therefore we have to measure L (solve the rendering equation) for every single pixel $m_1, m_2, \cdots, m_M$ in the final image, M being the total number of pixels in the image [33]. Each measurement will have the below form

$$m_j = \int_{S \times S} W^j{}_e(x \to x') L(x \to x') G(x \leftrightarrow x') dA(x) dA(x') \tag{3.14}$$

where $W^j{}_e(x \to x')$ is a weight that indicates how much the light arriving at x' from the direction of x contributes to the value of the measurement; $W^j{}_e(x \to x')$ in computer graphics is called importance function and in physics is called flux responsivity [33].

As it was mentioned, we need to define a function f as well as a measure μ with some constraints; now considering the above equation (measurement of L for each pixel) the goal is to rewrite that equation in form of f,

$$m_j = \int_{\Omega} f_j(\bar{x}) d\mu(\bar{x}) \tag{3.15}$$

Now (3.15) can be handled as a pure integration problem; we can also extract f and μ from

it:

$$f_j(x, x') = L_e(x \to x')G(x \leftrightarrow x')W^j{}_e(x \to x') \tag{3.16}$$

$$d\mu_k(x_0 \cdots x_k) = dA(x_0) \cdots dA(x_k) \tag{3.17}$$

If we let Ω be the union of all the Ω_k, and define a measure μ on Ω by the following equation:

$$\mu(D) = \sum_{k=1}^{\infty} \mu_k(D \cap \Omega_k) \tag{3.18}$$

Now we have an integral equation that can be approximated using Monte Carlo integration method. Another important topic that needs to be discussed is choosing the right mutation strategy. We can summarize the mutation strategies into the following cases (ξ denotes a random variable) [27],

- Applying random perturbations to the current sample [Equation]. For example one possibility is to perturb by adding or subtracting a scaled random variable:

$$x_{i+1} = x_i \pm s\xi \tag{3.19}$$

- Perturb using values from an exponential distribution; for example the following perturbation generates an exponentially distributed sample in the range $[a, b]$:

$$x_{i+1} = x_i \pm be^{-\log \frac{b}{a}}\xi \tag{3.20}$$

- Discard the current sample and generate a new one with uniform random numbers:

$$x_{i+1} = \xi \tag{3.21}$$

MLT is an unbiased algorithm, it handles general geometric and scattering models, uses little storage, and can be orders of magnitudes faster than previous unbiased algorithms;

it performs especially well on the problems that are usually considered difficult [33]. It is known for its robustness, meaning that while it is (at least) as efficient as other unbiased algorithms (e.g. bidirectional path tracing) in relatively simple scenes, it is considerably more efficient in dealing with scenes with difficult settings where most of the light transport happens along a small fraction of all of the possible paths through the scene [13, 27]. In general scenes with regions with bright direct light, non-diffuse reflections, caustics, glossy surfaces, and strong indirect lightings are considered difficult rendering problems and MLT in general is capable of handling all of these situations [33].

Another key advantage of MLT with respect to previous unbiased approaches for image synthesis is that it performs local exploration by favoring mutations that make small changes to the current path [33]. Most important consequences of local exploration are the average cost per sample is small (usually one or two rays per sample), when a path that makes a large contribution to the image is found, the nearby paths are also explored, therefore the cost of finding every single light transport path in the scene is reduced significantly; this will make it easy to sample other paths that are similar to the important paths by making small perturbations to them, and finally, the mutation set is easily extended by constructing mutations that preserve certain properties of the path while changing others.

However MLT is not perfect in every situation; for instance caustics due to mirror reflections are still difficult with MLT and especially in the case of point light sources it can't render this kind of scenes. Another difficulty with MLT is that picking the right mutation strategy is highly scene-dependent, and it can result in high variance in the final image [13].

3.6 Progressive Photon Mapping

As with photon mapping, the most notable advantage of PPM compared to unbiased Monte Carlo ray tracing algorithms such as path tracing, bidirectional path tracing, and Metropolis light transport is the ability to deal with complex caustics and the SDS path problem which

was discussed in Section 3.4.

Progressive photon mapping is a multi-pass rendering algorithm based on photon mapping, which starts with a single ray tracing pass, and then continues with running consecutive photon tracing passes to iteratively increase the accuracy of the estimate computation from previous passes. As the name suggests, the rendering process is progressive, meaning after each pass the algorithm produces visible results but as we continue with more passes the result gets more and more close to the accurate result.

The algorithm starts by running a standard ray tracing pass to find all the visible points in the scene from camera; a ray is reflected as long as it hits specular surfaces and stops when a non-specular surface is hit. If the scene has a large number of surfaces with specular components, Russian roulette can be used to terminate the ray. As a ray hits the surfaces in the scene, we store the hit points on the surfaces with non-specular components in their BRDFs; besides the point's position, we store a number of accompanying information such as ray direction $\vec{\omega}$, BRDF, associated pixel location, current photon radius (decreases as we run each photon tracing path), total unnormalized flux, etc.

After finishing the first path, the algorithm starts running consecutive photon tracing paths to increase the accuracy of the estimate by accumulating photons at each hit point found in the first step. Each pass proceeds by emitting some number of photons into the scene from the light sources and constructs a temporary photon map. The main advantage of PPM over the original photon mapping is that after a photon tracing pass is finished, there is no need to keep the photons in the map and they are discarded. These passes can continue indefinitely until we are satisfied with the estimate meaning we can use infinite number of photons without worrying about memory limitations (unlike photon mapping), or we can set a maximum number of total photons to be taken into account and after reaching that limit we can terminate the rendering process.

Progressive photon mapping uses progressive radiance estimate, a method introduced in the same paper, to increase the accuracy of the estimate at each photon tracing pass by

reducing the radius of the sphere and increasing the accumulated photons at point x.

Remember that in the first pass (standard ray tracing), we allocated a current photon radius to each stored hit point x, let's assume this radius is a function of x and denote it by $R(x)$, another piece of information accompanying x is the number of accumulated photons at x, denoted by $N(x)$. The goal is to decrease $R(x)$ by $dR(x)$ to get the new radius $R'(x)$, and at the same time increase $N(x)$ by some number of photons $M(x)$ in order to converge to the correct result. It is proven in [11] that the new radius for a pass can be given by:

$$R'(x) = R(x)\sqrt{\frac{N(x) + \alpha M(x)}{N(x) + M(x)}} \tag{3.22}$$

where $\alpha = (0, 1)$ chosen such that total amount of photons at x after the reduction in sphere's radius, is larger than it was after the previous photon tracing pass completed.

Now we need to calculate the accumulated flux at point x after increasing the number of photons and decreasing the radius. First step is to update the the value of total unnormalized flux for point x, denoted here by $\tau(x, \bar{\omega})$:

$$\tau(x, \omega) = \sum_{p=1}^{N(x)} f_r(x, \omega, \omega_p)\phi'_p(x_p, \omega_p) \tag{3.23}$$

where ω denotes the direction of incident ray at x, ω_p is the direction of incident photon, and $\phi'_p(x_p, \omega_p)$ is the flux carried by the photon p. To add the contribution of a new photon tracing pass, considering the reduced radius $R'(x)$ and the addition of $\alpha M(x)$ photons originally, and $\hat{N}$ photons in total after the radius reduction, it is proven that the new increased total flux is:

$$\tau_{\hat{N}}(x, \omega) = \tau_{N+M}(x, \omega)\frac{N(x) + \alpha M(x)}{N(x) + M(x)} \tag{3.24}$$

where $\tau_{N+M} = \tau_N + \tau_M$ (which would not need the scaling factor if the radius remained constant).

At this point the algorithm can proceed to estimate the radiance at point x for the current

photon tracing pass, which can be calculate using:

$$L(x,\omega) \approx \frac{1}{\pi R(x)^2} \frac{\tau(x,\omega)}{N_{emitted}} \tag{3.25}$$

where $N_{emitted}$ is the total number of emitted photons so far. Note that the algorithm assumes the photon density and illumination inside the disc is constant.

4. Sample Results and Comparison of MCRT Algorithms

We presented an overview of some of the most important and highly cited Monte Carlo ray tracing algorithm in Chapter 3, including:

1. Path tracing (PT) [20],
2. Bidirectional path tracing (BPT) [23, 24, 31, 32],
3. Photon mapping (PM) [15, 16, 14, 17],
4. Metropolis light transport (MLT) [33], and
5. Progressive photon mapping (PPM) [11].

To better understand the differences between these algorithms and their advantages and disadvantages, it is important to compare them based on the images they produce using different scene settings. For this purpose, we have employed a research-oriented rendering system named Mitsuba developed and maintained by Wenzel Jakob.[1]

We will examine sample results for each of the algorithms discussed in Chapter 3 and compare their performance both in terms of rendering time and accuracy (e.g., amount of noise) on model scenes widely used throughout the rendering literature to assess the performance of rendering algorithms. For all cases the Mitsuba renderer was run on an 8 logical core Intel Core i7 computer clocked at 2.6 GHz.

Three different scenes were selected to perform the tests. First we use the "Cornell Box" scene which contains only diffuse surfaces, to assess the performance and accuracy of the

[1] Website: https://www.mitsuba-renderer.org/

above algorithms in such scenes. Next we will use the Torus Scene (from [6]) to assess performance and accuracy in the context of SDS path problems (complex caustics effect which were discussed in Section 3.4). Finally, one of the scenes from [33] is used to assess the performance in the situation described in Figure 3.1 where the primary source of light in a scene is indirect lighting.

As demonstrated in Figure 4.1, taking more samples results in a decrease in noise. Also note that MLT needs more samples to converge to the correct result, but at the same time examining the rendering times in Table 4.1, it performs better than PT and much better than BPT in terms of rendering time. Table 4.1 also demonstrates that in terms of rendering time and efficiency, BPT performs worse than PT and MLT, and MLT performs better than the other two algorithms. These results show that, taking both accuracy and rendering time into consideration, MLT is the best approach among these three algorithms for scenes consisting entirely of diffuse surfaces.

The second scene consists of a torus with diffuse surfaces located completely inside a glass cube in order to test each of these algorithms for the SDS path problem. Figure 4.2 is the result of rendering this scene with PT, BPT, and MLT with an increasing number of samples per pixel. Figure 4.2 clearly shows that, among these unbiased MC methods, only MLT is capable of dealing with this scene, while PT and BPT get worse as they take more samples. Rendering times are tabulated in Table 4.2.

The third scene demonstrates the problem of occluded light sources and indirect illumination of the scene (refer to Section 3.3). As it is demonstrated in Figure 4.3 and mentioned in Section 3.3, PT cannot efficiently deal with these types of scenes. Not only have caustics effect in this scene, but the primary light source is occluded and the scene is illuminated indirectly by the walls and the ceiling. Here MLT again outperforms PT and BPT when both rendering time and accuracy are taken into account. Table 4.3 provides the rendering times.

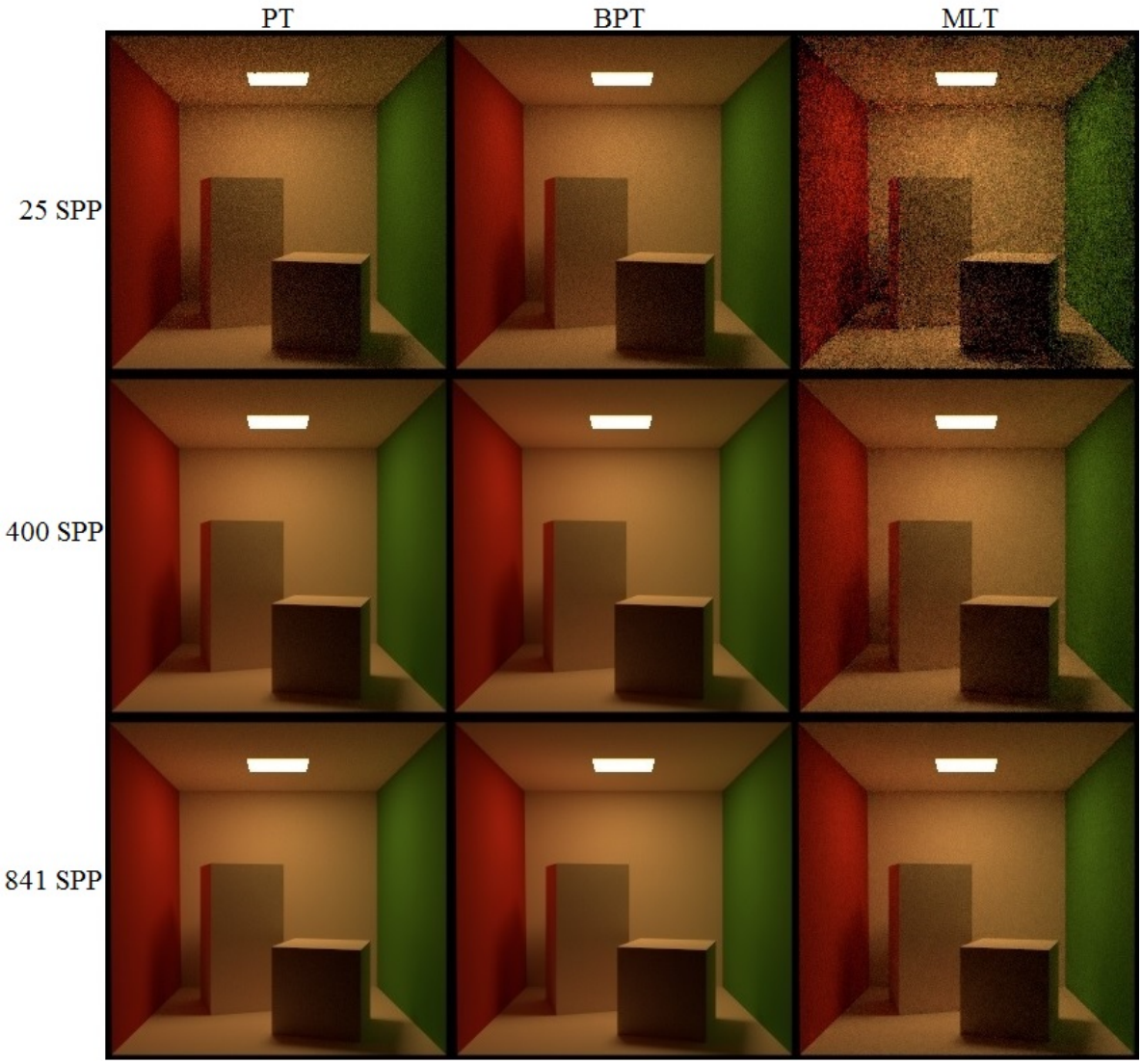

Figure 4.1: PT, BPT, and MLT results as the number of samples taken per pixel increases. Note that in this example MLT needs more samples to converge to better estimates compared to PT and BPT, but at the same time its performance is better than PT and much better than BPT. Refer to Table 4.1 for rendering times.

Cornell Box rendering (256×256) times in seconds (unbiased algorithms)			
Algorithm	25 SPP	400 SPP	841 SPP
PT	1s	17s	37s
BPT	6s	92s	200s
MLT	3s	14s	27s

Table 4.1: Rendering times for results in Figure 4.1 (Cornell Box scene).

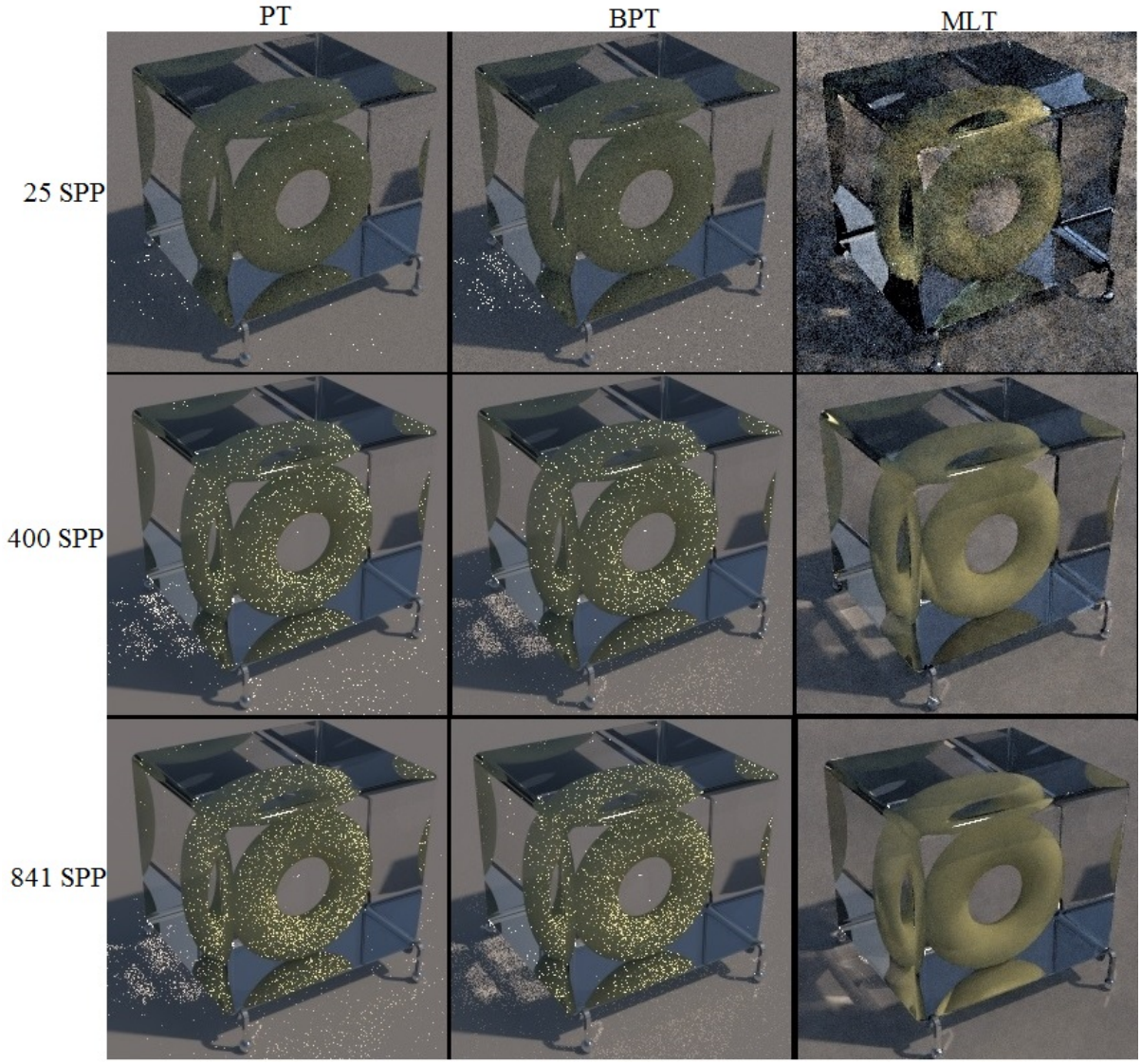

Figure 4.2: This scene demonstrates the problem of finding SDS paths by unbiased Monte Carlo rendering algorithms. Note that PT and BPT degrade as they use more samples, since these algorithms cannot find the correct path; therefore, paths with little contribution are taken into account instead of the correct paths of SDS form, hence the noise in the results.

Torus scene [6] rendering (256×256) times in seconds (unbiased algorithms)			
Algorithm	25 SPP	400 SPP	841 SPP
PT	2s	22s	47s
BPT	3s	49s	109s
MLT	3s	33s	69s

Table 4.2: Rendering times for results in Figure 4.2 (torus scene with caustics).

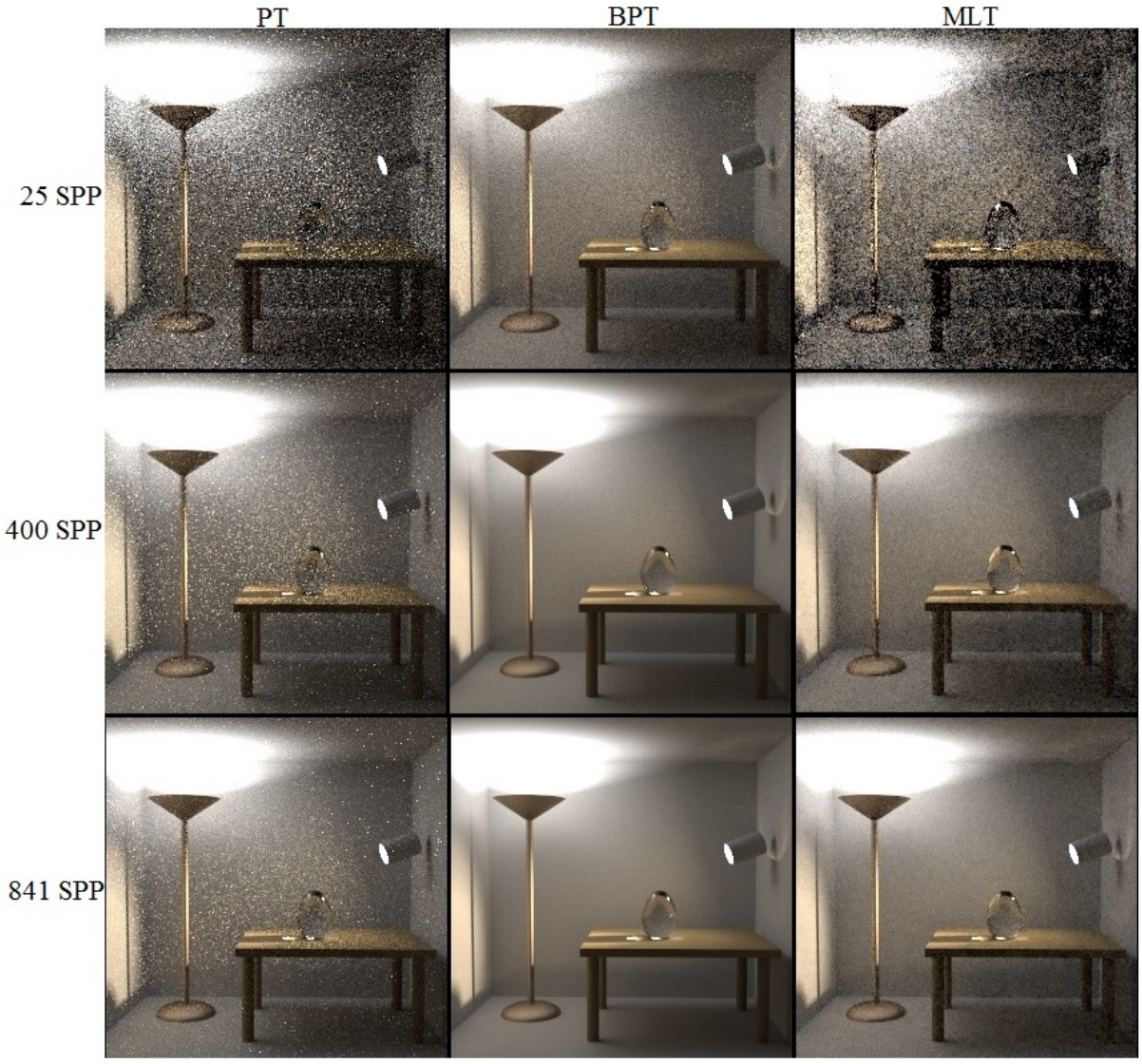

Figure 4.3: This set of images demonstrate the problem of tracing paths to an occluded light source (indirect illumination by occluded light source) in PT. On the other hand, BPT and MLT can converge to the correct result quickly by taking more samples. Also note that the caustics effect in this scene is different from the SDS problem. Hence, all three algorithms seem to be capable of handling it reasonably well.

Room scene [33] rendering (256×256) times in seconds (unbiased algorithms)			
Algorithm	25 SPP	400 SPP	841 SPP
PT	2s	31s	67s
BPT	8s	138s	294s
MLT	4s	20s	38s

Table 4.3: Rendering times for results in Figure 4.3 (room scene with indirect illumination).

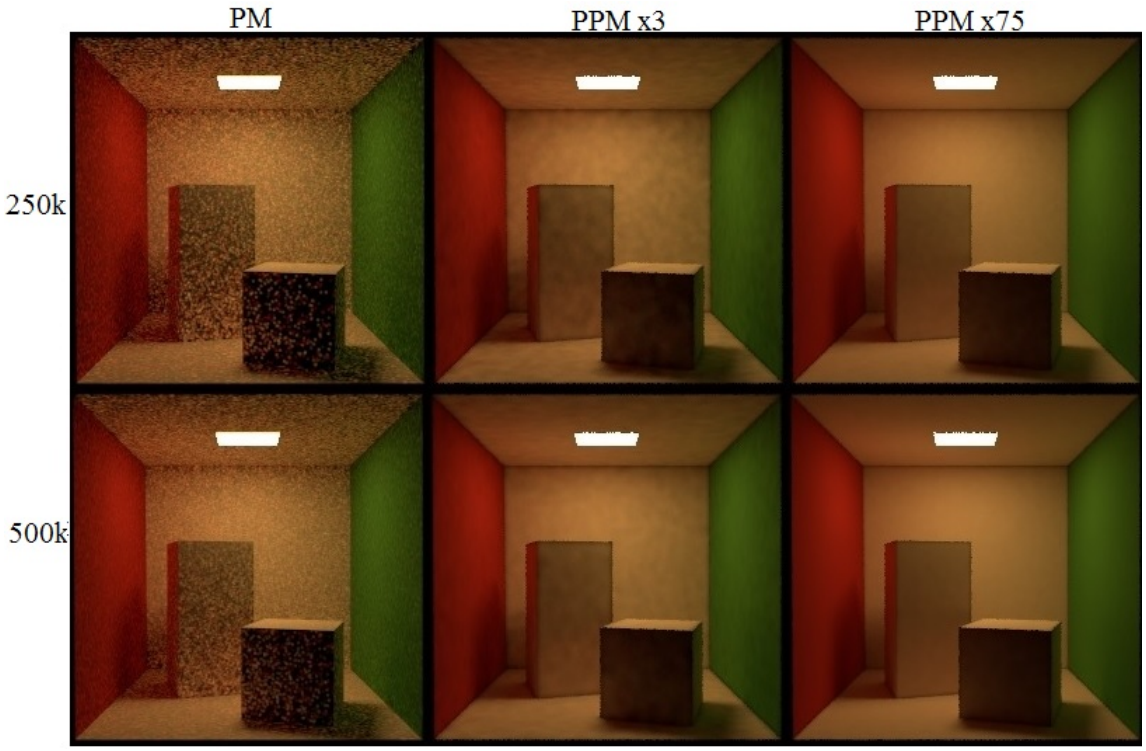

Figure 4.4: PM and PPM results improve as the number of photons emitted into the scene is increased for PM, and as we perform more photon tracing passes in PPM. Refer to Table 4.4 for rendering times.

Cornell Box rendering (256×256) times in seconds (biased algorithms)		
Algorithm	250k photons/pass	500k photons/pass
PM	53s	4s
PPM (3 passes)	2s	56s
PPM (75 passes)	29s	52s

Table 4.4: Rendering times for results in Figure 4.4 (Cornell box scene).

As the results so far demonstrate, Metropolis Light Transport outperforms both Path Tracing and Bidirectional Path Tracing and is capable of dealing with all the shortcomings of other unbiased rendering algorithms; therefore we can argue that MLT is the best choice among the unbiased rendering algorithms we discussed here, both in terms of performance and robustness.

Now we will proceed to examine the results for the biased algorithms we discussed in Chapter 3 for the same set of test scenes.

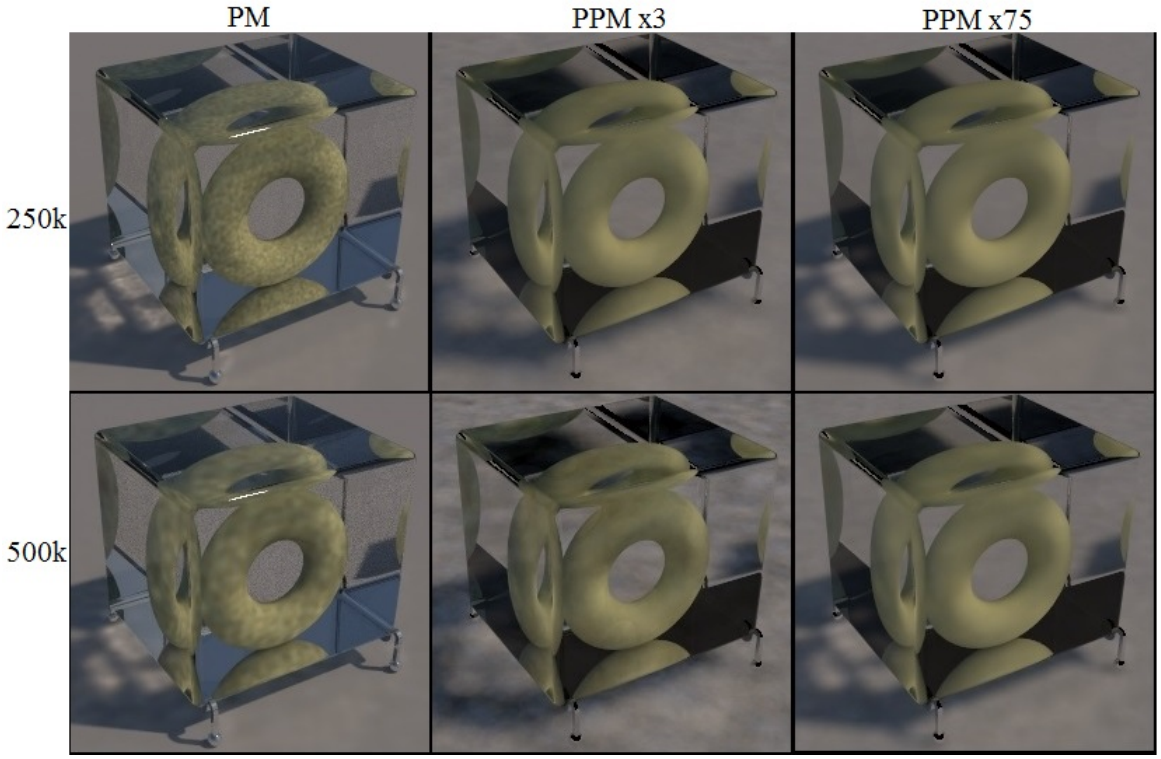

Figure 4.5: This scene demonstrates the most important advantage of the biased rendering algorithms we discussed. As the results show, unlike the unbiased algorithms, PM and PPM have no trouble converging to a good estimate for the caustics in this scene.

Torus scene [6] rendering (256×256) times in seconds (biased algorithms)		
Algorithm	250k photons/pass	500k photons/pass
PM	54s	61s
PPM (3 passes)	22s	58s
PPM (75 passes)	222s	438s

Table 4.5: Rendering times for results in Figure 4.5 (torus scene with caustics).

Room scene [33] rendering (256×256) times in seconds (biased algorithms)		
Algorithm	250k photons/pass	500k photons/pass
PM	38s	76s
PPM (3 passes)	2s	4s
PPM (75 passes)	35s	71s

Table 4.6: Rendering times for results in Figure 4.6 (room scene with indirect illumination).

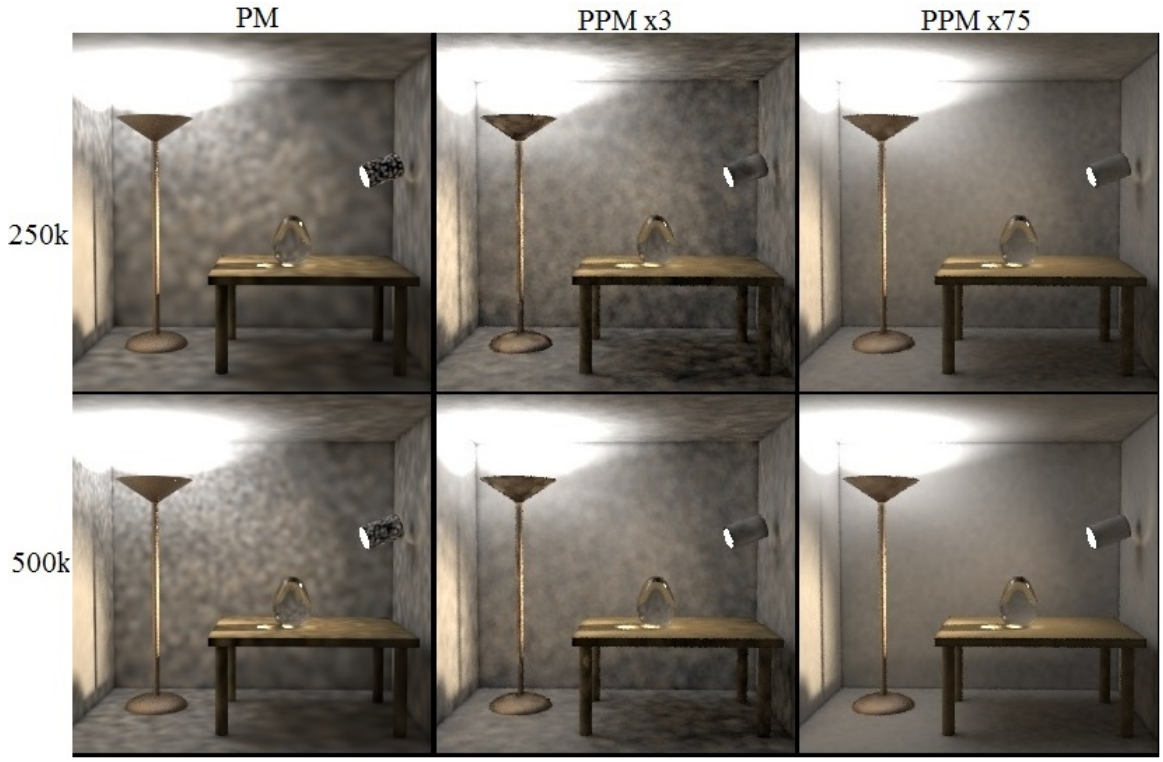

Figure 4.6: Here the problem with PM is only lack of a sufficient number of photons stored in the photon map, and PPM is reaching a good estimate as more photon tracing passes are completed.

5. Conclusion

This thesis reported on our investigation of the problem of global illumination and photorealistic rendering using Monte Carlo integration techniques. The focus was mostly on the classical algorithms such as path tracing (PT), bidirectional path tracing (BPT), photon mapping, and Metropolis light transport. These four algorithms are arguably among the most important contributions to the rendering literature, and they are without a doubt the most influential algorithms in the field of photorealistic rendering, consistently cited in the literature till today. Most of what researchers have introduced in the past two decades or so has been incremental improvements of these classical algorithms.

On the basis of our comparative evaluation reported in the previous chapter, we can conclude overall that except for the SDS path problem, Metropolis light transport is still an efficient and robust unbiased Monte Carlo rendering algorithm; not only does it converge to the correct results usually faster than the alternatives, but the results are not as noisy as the PT and BPT results. On the other hand, Progressive Photon Mapping seems to be an algorithm that can yield good results in general, including for SDS path scenarios by performing more photon tracing passes.

5.1 Outlook

Given the direction that the entertainment industry, both the motion picture and video game industries, has been pursuing in the past few years, it is not hard to see the current trend of moving towards ray-tracing-based algorithms and slowly abandoning the ad-hoc realtime rendering techniques which are common place today. This is true even for interactive applications such as virtual reality and video games for which there are two great examples—

Unreal Engine 4[1] and the Brigade Renderer.[2]

It is also worth noting that although animation production studios have been using some form of ray tracing to render specific effects that are hard or expensive to solve using rasterization-based methods, the trend of moving towards Monte Carlo ray tracing and physically-based rendering is fairly recent.

Another driver behind the new found interest in these decades old algorithms in recent years is the advances in hardware, such as multicore processors, GPUs with thousands of processing units, as well as the availability of fast and cheap memory. The advances in hardware have been so dramatic that these past few years at the ACM SIGGRAPH conference we can see hardware manufacturers such as NVIDIA and Intel, rather than focusing on the current rasterization-based graphics pipelines, starting to talk about the next generations of graphics hardware based on ray tracing.[3]

[1] Epic Games, Inc.

[2] OTOY, Inc.

[3] For instance at SIGGRAPH 2013 a course was offered entitled "Ray Tracing is the Future and Ever Will Be" discussing this issue.

6. Bibliography

[1] Amanatides, J. 1984. Ray tracing with cones. *ACM SIGGRAPH Computer Graphics 18*, 3, 129-135.

[2] Appel, A. 1968. Some techniques for shading machine renderings of solids. *Proceedings of the April 30–May 2*, 1968, Spring Joint Computer Conference (AFIPS '68 (Spring)), 37-45.

[3] Arvo, J. 1986. Backward ray tracing. *ACM SIGGRAPH '86 Course Notes — Developments in Ray Tracing.*

[4] Bekaert, P., Sbert, M. and Halton, J. 2002. Accelerating path tracing by re-using paths. *Proceedings of the 13th Eurographics Workshop on Rendering (EGRW '02)*, 125-134.

[5] Clark, J. 1976. Hierarchical geometric models for visible-surface algorithms. *ACM SIGGRAPH Computer Graphics 10*, 2, 267-267.

[6] Cline, D., Talbot, J. and Egbert, P. 2005. Energy redistribution path tracing. *ACM Transactions on Graphics 24*, 3, 1186.

[7] Cook, R. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1, 51-72.

[8] Cook, R., Porter, T. and Carpenter, L. 1984. Distributed ray tracing. *ACM SIGGRAPH Computer Graphics 18*, 3, 137-145.

[9] Goral, C., Torrance, K., Greenberg, D. and Battaile, B. 1984. Modeling the interaction of light between diffuse surfaces. *ACM SIGGRAPH Computer Graphics 18*, 3, 213-222.

[10] Gregory, J. 2014. *Game Engine Architecture, Second Edition.* CRC Press, Hoboken.

[11] Hachisuka, T., Ogaki, S. and Jensen, H. 2008. Progressive photon mapping. *ACM Transactions on Graphics 27*, 5, 1.

[12] Heckbert, P. 1990. Adaptive radiosity textures for bidirectional ray tracing. *ACM SIGGRAPH Computer Graphics 24*, 4, 145-154.

[13] Jensen, H. 2001. *Realistic image synthesis using photon mapping*. A K Peters, Natick, MA.

[14] Jensen, H. 1995. Importance driven path tracing using the photon map. In: P. Hanrahan and W. Purgathofer, ed., *Rendering Techniques '95*. Springer Vienna, 326-335.

[15] Jensen, H. 1996. Global illumination using photon maps. In: X. Pueyo and P. Schröder, ed., *Rendering Techniques '96*. Springer-Verlag, 21-30.

[16] Jensen, H. and Christensen, N. 1995. Photon maps in bidirectional Monte Carlo ray tracing of complex objects. *Computers & Graphics 19*, 2, 215-224.

[17] Jensen, H. and Christensen, N. 1995. Efficiently rendering shadows using the photon map. *Proceedings of Compugraphics '95*, 285-291.

[18] Jensen, H. and Christensen, N. 1995. Optimizing path tracing using noise reduction filters. *Proceedings of WSCG 1995*, 134-142.

[19] Jensen, H., Arvo, J. and Dutré, P. et al. 2003. Monte Carlo ray tracing. *ACM SIGGRAPH 2003 Courses (SIGGRAPH '03)*.

[20] Kajiya, J. 1986. The rendering equation. *ACM SIGGRAPH Computer Graphics 20*, 4, 143-150.

[21] Keller, A. 2013. Quasi-Monte Carlo image synthesis in a nutshell. *Springer Proceedings in Mathematics and Statistics 65*.

[22] Keller, A., Premoze, S. and Raab, M. 2012. Advanced (quasi) Monte Carlo methods for image synthesis. *ACM SIGGRAPH 2012 Courses (SIGGRAPH '12)*.

[23] Lafortune, E. and Willems, Y. 1993. Bidirectional path tracing. *Proceedings of the Third International Conference on Computational Graphics and Visualization Techniques*, 145-153.

[24] Lafortune, E. and Willems, Y. 1994. A theoretical framework for physically based rendering. *Computer Graphics Forum 13*, 2, 97-107.

[25] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. 1953. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics 21*, 6, 1087.

[26] Moller, T. and Haines, E. 2002. *Real-time rendering*. AK Peters, Natick, Mass.

[27] Pharr, M. and Humphreys, G. 2010. *Physically based rendering*. Morgan Kaufmann, San Francisco, Calif.

[28] Roth, S. 1982. Ray casting for modeling solids. *Computer Graphics and Image Processing 18*, 2, 109-144.

[29] Shirley, P., Edwards, D. and Boulos, S. 2008. Monte Carlo and quasi-Monte Carlo methods for computer graphics. In: *A. Keller, S. Heinrich and H. Niederreiter, ed., Monte Carlo and Quasi-Monte Carlo Methods 2006*. Springer Berlin Heidelberg.

[30] Shirley, P., Wade, B., Hubbard, P., Zareski, D., Walter, B. and Greenberg, D. 1995. Global illumination via density estimation. *Proceedings of 6th Workshop on Rendering*, 219-230.

[31] Veach, E. and Guibas, L. 1994. Bidirectional estimators for light transport. *Eurographics Rendering Workshop 1994 Proceedings*, 147-162.

[32] Veach, E. and Guibas, L. 1995. Optimally combining sampling techniques for Monte Carlo rendering. *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95).*

[33] Veach, E. and Guibas, L. 1997. Metropolis light transport. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH '97)*, 65-76.

[34] Waldman, G. 1983. *Introduction to light.* Prentice-Hall, Englewood Cliffs, N.J.

[35] Ward, G., Rubinstein, F. and Clear, R. 1988. A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics 22*, 4, 85-92.

[36] Whitted, T. 1979. An improved illumination model for shaded display. *ACM SIGGRAPH Computer Graphics 13*, 2, 14.