Extensions to Bidirectional Path Tracing

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

Computer Science

by

Aleksandr Palatnik

Committee in charge:

Professor Henrik Jensen, Chair
Professor Sam Buss
Professor Geoffrey Voelker

2014
The Thesis of Aleksandr Palatnik is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2014
DEDICATION

Thanks to my parents, for supporting me the whole way.
# TABLE OF CONTENTS

Signature Page .......................................................... iii
Dedication ................................................................. iv
Table of Contents ....................................................... v
List of Figures ........................................................... vii
List of Tables ........................................................... viii
Acknowledgements ...................................................... ix
Abstract of the Thesis ................................................. x

Chapter 1  Introduction ................................................ 1
  1.1 Motivation/Goals ............................................. 1
  1.2 Context ....................................................... 2
  1.3 Approach ..................................................... 3
  1.4 Overview ..................................................... 4

Chapter 2  Rendering .................................................. 5
  2.1 Radiometry ................................................... 6
  2.2 Local Illumination .......................................... 8
  2.3 The Rendering Equation .................................... 10
    2.3.1 Importance ........................................... 13

Chapter 3  Monte Carlo Integration ................................. 15
  3.1 Basic Monte Carlo Estimator ............................... 15
  3.2 Monte Carlo Error .......................................... 17
    3.2.1 Bias .................................................. 17
    3.2.2 Consistency ......................................... 18
  3.3 Variance Reduction ......................................... 18
    3.3.1 Importance Sampling ............................... 19
    3.3.2 Multiple Importance Sampling ..................... 20
  3.4 Monte Carlo Rendering ..................................... 22
    3.4.1 Performance Improvements ......................... 23

Chapter 4  Rendering Algorithms .................................. 26
  4.1 Path Tracers ................................................ 27
    4.1.1 Basic Path Tracing ................................ 27
  4.2 Bidirectional Path Tracing ................................ 33
    4.2.1 Path Integral Formulation ......................... 33
LIST OF FIGURES

Figure 1.1: A bidirectional path tracing technique utilizing two light vertices and three eye vertices. .......................................................... 3

Figure 2.1: Surface patch $dA$ at point $\mathbf{x}$, with normal $\mathbf{n}_x$. Radiance is the total flux arriving at $dA$ from the differential solid angle $d\omega$, adjusted for foreshortening. ................................................................. 7

Figure 2.2: A graphical depiction of the values involved in radiance transfer between two surface patches. ......................................................... 7

Figure 2.3: Integration over the unit hemisphere. The hemispherical rendering equation gives the amount of power exiting point $\mathbf{x}$ along a single direction $\omega_o$ based on an integration over all incoming directions $\omega_i$. 11

Figure 4.1: The first few points and directions involved in a single recursive path evaluation. ................................................................. 29

Figure 4.2: Taking 500 samples per pixel and 161 seconds of computation time, path tracing is still far from converged caustics in a relatively simple scene. ................................................................. 30

Figure 4.3: Path tracing next event estimation: explicit direct lighting contributions at each bounce in the traced path. ................................................................. 31

Figure 4.4: The six bidirectional techniques for generating a path of length $k = 4$. 38

Figure 4.5: The four possible prefix-suffix connections of length one subpaths. 39

Figure 4.6: Projection of a vector onto a simulated pinhole camera image plane. 47

Figure 4.7: Equal computation time comparison between path tracing with next event estimation and bidirectional path tracing. 49

Figure 5.1: Multipath bidirectional path tracing with $n_l = 2, n_e = 1$ produces two explicit edges for the $s = 2, t = 1$ technique. 51

Figure 5.2: Reference renders were performed using standard bidirectional path tracing techniques with a very high number of samples per pixel, and are considered sufficiently-converged for testing purposes. Both images required over an hour of render time on the test machine. 53

Figure 5.3: Efficiency curves showing reduction of error with render time of BDPT versus different settings of MBDPT. 58

Figure 5.4: Branching bidirectional path tracing uses trees instead of paths, so different techniques can used to create varying numbers of samples. This is an illustration of multiple $s = 3, t = 4$ technique samples. 60

Figure A.1: Example cumulative density function for the PDF $p_X(x) = \frac{1}{\sqrt{2}}$, and an illustration of how 1D inversion works. 64

Figure A.2: Illustration for specular bounce example. 77


LIST OF TABLES

Table 5.1: Indirect Scene equal-time rendering results . . . . . . . . . . . . . . 54
Table 5.2: Hard Scene equal-time rendering results . . . . . . . . . . . . . . 55
Table A.1: Combinations of specular and diffuse components in cameras. . . . . 79
ACKNOWLEDGEMENTS

I would like to acknowledge Professor Henrik Jensen, Professor Alan Barr, and Luke Durant, for teaching me most of what I know about computer graphics, and Eric Veach, from whose thesis I learned the rest.
ABSTRACT OF THE THESIS

Extensions to Bidirectional Path Tracing

by

Aleksandr Palatnik

Master of Science in Computer Science

University of California, San Diego, 2014

Professor Henrik Jensen, Chair

Bidirectional path tracing is a fairly well-understood and reliable rendering algorithm, though beyond guiding its sample selection method, little has been done in exploring different sample geometry. This thesis aims to explore the possibility of using more complex path geometry to gain more data from a limited amount of precomputed data. As a secondary goal, this thesis is written to be an introductory primer to unbiased offline rendering and bidirectional path tracing.

In this thesis, I describe the fundamental underpinnings of bidirectional path tracing, including radiometry, Monte Carlo integration methods, precursor algorithms; and develop a new extension to bidirectional path tracing. This extension, multipath bidirectional path tracing, creates multiple light and eye subpaths for each sample in order to reuse more precomputed vertex data. I also discuss a second potential extension,
branching bidirectional path tracing, which creates eye and light trees instead of paths, where the level of branching at each surface intersection can be controlled at runtime, allowing for another avenue for importance sampling.
Chapter 1

Introduction

Rendering is a very open field of research, and even well-established algorithms haven’t explored many potential optimizations. One such algorithm, bidirectional path tracing, is able to effectively render many lighting phenomena, but like any Monte Carlo rendering algorithm, it can take a considerable amount of time to do so. This thesis discusses a modification to the way this algorithm gathers and combines samples based on an older algorithm, distributed ray tracing\(^1\). The goal is to gain greater useful information per sample, improving the algorithm’s efficiency. The first few chapters of this thesis are also intended to be an introduction to the mathematics and techniques behind physically-based Monte Carlo rendering.

1.1 Motivation/Goals

Physically-based global illumination rendering is a long-standing problem, with applications in architecture, art, animation, and simulation, but in a sense can be thought of as a solved problem. At its most basic level, this is a problem of simulating the interaction of light with materials, and physics has given us precise mathematical descriptions of this behavior, even to the point of explaining effects we generally never perceive. Given the necessary equations, numerical solutions for any scene can be found. Even simple path tracing, an algorithm conceived in 1986 by James Kajiya\(^2\), will produce a

\(^1\)See [CPC84].
\(^2\)See [Kaj86].
correct result if given enough time. Despite this, research continues.

The real problem is efficiency. Path tracing can produce perfect results in even the most complex scenes, but we are generally not willing to wait the years it would require to do so. Even with modern advances in computers, from faster clock rates to massively-parallel processors, physically-based global illumination algorithms often operate very slowly, forcing users to choose between accuracy and computation time. Many rendering algorithms and optimizations to remedy this have been designed since the days of simple ray tracing, using new methods of simulating the same physical principles in pursuit of speed and capability. This thesis will not go into the details of the most modern of these techniques, opting instead to focus on a relatively simple algorithm, bidirectional path tracing. While this algorithm has some complex subtleties, once those are understood, its implementation and modification is straightforward. Without going into excessive detail here, many methods of optimizing this algorithm have been devised, but very little has been done to explore distribution, an older technique in ray tracing. While its benefit to unidirectional algorithms was limited, this thesis aims to determine whether distribution can be significantly improve bidirectional path tracing.

This text also has a secondary goal. Though many different rendering algorithms have been devised, they very often share the same fundamental mathematical underpinnings. Many texts exist that explain these concepts, but few seem to achieve a good balance mathematical correctness, implementation discussion, and clarity, often leaning too heavily on one at the expense of the others. This thesis aims to provide a clearer explanation of these fundamentals and provide a basic, yet thorough, primer on physically-based global illumination.

1.2 Context

Distributed ray tracing is an older algorithm designed in 1984\(^3\) and is generally abandoned because its benefits depend greatly on the scene being rendered. The basic approach of distributed ray tracing is the same as ordinary path tracing: fire rays from

\(^3\)See [CPC84].
a camera into a scene, then bounce them around depending on material properties. The
difference is that at each bounce, if the reflected direction of the ray is not deterministic,
the algorithm fires multiple reflected rays and averages their resulting contributions.
This can reduce noise due to diffuse surfaces, but may also compute many unneeded
additional rays, depending on the arrangement of the scene being rendered.

While ultimately distribution proved to be impractical and overly-dependent on
the scene being rendered, this thesis aims to test if a similar approach, namely the com-
bination of additional samples per pass, can work more beneficially for bidirectional
path tracing. In its most pure form, bidirectional path tracing traces two subpaths with
randomized lengths, one each for the camera and a light source, and connects their
endpoints, creating a complete contributing path. However, typically this algorithm is
implemented such that the contribution data is precomputed at every bounce in the two
subpaths and all pairs of bounces (one from each subpath) are connected, providing
multiple contributions. Because of this precomputation, it is possible that multiple sam-
ples per pass\(^4\) can create a tangible benefit simply by producing more potential paths to
evaluate.

1.3 Approach

\[\text{Figure 1.1: A bidirectional path tracing technique utilizing two light vertices and three eye vertices.}\]

The algorithm described in this thesis, multipath bidirectional path tracing, does
not directly implement the branching nature of distributed ray tracing. Instead, it is
based on the general idea of using multiple samples of the same behavior and combin-

\(\text{\(^4\)A pass is the creation and connection vertices along eye and light paths. The samples are the paths themselves.}\)
ing them. Bidirectional path tracing as described by Eric Veach only gathers one sample from each "technique" (a specific selection of eye and light path lengths, as seen in figure 1.1), combining all samples using a weighted average. Multipath bidirectional path tracing, or MBDPT, instead precomputes multiple eye and light paths, and tests connections between all pairs, thus creating multiple samples from each technique. Using multiple paths provides a multiplicative increase in the number of samples with only an additive increase in the amount of precomputed data\(^5\). Naturally, these samples are very correlated since they are all generated from the same sets of path vertices, and as such increase the variance in the results, but this has a minimal effect on the results of normal bidirectional path tracing\(^6\), so this thesis aims to investigate whether the sampling gains from MBDPT overcome the additional correlation noise.

### 1.4 Overview

This thesis is organized as follows. Chapter 2 covers radiometry and basic rendering concepts and equations. Chapter 3 discusses Monte Carlo integration, to a large degree without specificity to rendering. Chapter 4 covers several existing rendering algorithms, focusing on bidirectional path tracing and its basic optimizations. Chapter 5 details the new modifications to bidirectional path tracing, as well as additional proposed modifications for future work. Chapter 6 discusses the implications of the results presented here, and covers some potential avenues for future research.

\(^5\)For example, using two light paths and two eye paths requires twice the amount of precomputed data (one set for each path), but produces four times as many samples.

\(^6\)See [VG95, p.306].
Chapter 2

Rendering

A photorealistic renderer is, at a most basic level, a program that takes as input a scene containing geometry, light-emission information, and material descriptions, and produces a representation of the steady-state lighting of the scene as dictated by physical principles\(^1\). Depending on the algorithm, this representation can be a single image, set of images, or lighting information represented in the scene itself. The rendering algorithm solves the global illumination (or GI) problem by finding the physically accurate illumination at every point in the scene relevant to the output representation.

Using simple path tracing algorithms as an example, the solution to the GI problem consists of the light received by each pixel in the "sensor" of a camera somewhere in the scene. Radiosity algorithms, by contrast, partition all surfaces in the scene into patches, then calculate the steady-state illumination of each patch. Rather than produce a single image, these algorithms store the solution to the GI problem as reflected light information for each patch in the scene, allowing the result to be used for real-time rendering\(^2\).

Before discussing the details of rendering algorithms, this chapter will cover some of the fundamental mathematics and ideas used in physically-based rendering. Section one will cover basic radiometric quantities, the values used to actually measure

\(^1\)While these algorithms are based on physical principles, there are generally simplifying assumptions made since some physical principles are extremely hard to model. Common simplifications are to ignore quantum effects, fluorescence, and phosphorescence, though there are ways to model even these phenomena.

\(^2\)This comes at the cost of limited lighting effects (like a lack of mirrors), and a tradeoff between fine detail and storage requirements.
and describe lighting in a scene. Sections two and three will cover local illumination, the interaction of light with a point on a surface, and the rendering equation, the basis for global illumination renderers.

2.1 Radiometry

Most fundamentally, rendering is the quantification of light’s interaction with a given scene. As such, quantities describing the amount and directionality of emitted, absorbed, or transmitted energy must be defined.

The most basic unit, radiant power, or flux, is simply the amount of energy striking, leaving, or passing through a given surface at a given time. Measured in watts (W), it is denoted by the symbol $\Phi$. Flux is measured over the entire surface being considered, so for example, a 10x10 uniformly-bright square light putting out a total of 100 watts of energy has the same radiant power as a 1x1 square light putting out 100 watts. However, the apparent brightness of the two lights will differ since there is a much greater amount of light emitted by any given point on the smaller light.

In the case of an ideal diffuse surface, where the emitted or reflected light does not depend on direction, the difference in brightness of the two lights is described by the radiant exitance $M$ of the surface. Radiant exitance is the amount of flux leaving a surface per unit area (irradiance $E$, conversely, is the amount of flux arriving at a surface per unit area). Radiant exitance (also called radiosity, denoted $B$) and irradiance are both measured in $\text{W/m}^2$.

$$E = \frac{d\Phi}{dA}, \quad B = M = \frac{d\Phi}{dA}. \quad (2.1)$$

Returning to the light example, the larger light has a flux of 100 watts, and an area of 100 $\text{m}^2$, giving it a radiosity of 1 $\text{W/m}^2$. The smaller light appears brighter because it has a much smaller area, giving it a radiosity of 100 $\text{W/m}^2$. While both radiosity and irradiance can describe the lighting of any surface, they do not capture directionality, so they can completely characterize only diffuse surfaces.

To completely describe the light arriving at or leaving an arbitrary surface, radiance is used. Radiance $L$ is defined as the amount of flux arriving at (or leaving) a
Figure 2.1: Surface patch $dA$ at point $x$, with normal $\vec{n}_x$. Radiance is the total flux arriving at $dA$ from the differential solid angle $d\omega$, adjusted for foreshortening.

Figure 2.2: A graphical depiction of the values involved in radiance transfer between two surface patches.

Surface per unit projected area per solid angle. Radiance is defined as

$$L = \frac{d^2\Phi}{d\sigma dA^\perp} = \frac{d^2\Phi}{d\sigma dA \cos(\theta)}.$$  \hspace{1cm} (2.2)

The projection takes into account that given a direction of traveling light, the surface area apparent from that direction decreases as the surface tilts away from perpendicular. Radiance has the useful properties of being invariant over straight paths (in the absence of participating media), and being invariant with respect to the direction of light travelling between two surface patches.

To prove these properties, consider two small surface patches with areas $dA_1$ and $dA_2$ [Nic63]. At patch 1, the solid angle subtended by patch 2 is

$$d\sigma_1 = \frac{\cos \theta_2 dA_2}{r^2}$$  \hspace{1cm} (2.3)
where $\theta_2$ is the angle between the normal of patch 2 and the direction separating the two patches, and $r$ is the distance between them. Likewise, $d\sigma_2$ is the solid angle subtended by patch 1 at patch 2. These values are illustrated in figure 2.2. Now, considering a beam of light traveling between the two patches, the flux $d\Phi_1$ leaving patch 1 is equal to the flux $d\Phi_2$ arriving at patch 2 in the absence of an interfering (i.e. scattering) medium. Using the definition of radiance,

$$d\Phi_1 = L_1 \cos \theta_1 dA_1 d\sigma_1 = L_2 \cos \theta_2 dA_2 d\sigma_2 = d\Phi_2$$

(2.4)

where $L_1$ is the radiance exiting patch 1 towards patch 2, and $L_2$ is the incoming radiance at patch 2 from patch 1. Now, using the definitions of the subtended solid angles,

$$\cos \theta_1 dA_1 d\sigma_1 = \cos \theta_1 dA_1 \frac{\cos \theta_2 dA_2}{r^2} = \frac{\cos \theta_1 dA_1}{r^2} \cos \theta_2 dA_2 = \cos \theta_2 dA_2 d\sigma_2$$

(2.5)

which combined with equation 2.4 means that $L_1 = L_2$, showing that radiance is invariant along the beam connecting two surface patches. This also proves the directional invariance property since the choice of which patch is the emitter and which is the receiver is arbitrary, so switching the two will not affect the proof.

### 2.2 Local Illumination

While solving the global illumination problem is the end goal of a photorealistic renderer, the solution depends entirely on local illumination (or shading), the interaction of light with any medium in the scene. Local illumination is governed most generally by the bidirectional surface scattering reflectance distribution function (BSSRDF) of the material being considered. The BSSRDF describes the distribution of outgoing radiance at one surface point given the distribution of incoming radiance at a potentially different point on the same object. While the BSSRDF is the most accurate description of a homogeneous medium, most renderers use a bidirectional scattering distribution function (BSDF) instead. The BSDF also gives an outgoing radiance distribution given an incoming radiance distribution, but

---

3 This includes participating media, which won’t be considered for this thesis.
both are at the same point on the object surface. Using only a single point greatly simplifies the local illumination problem since there is no longer a need to consider the scene as a whole when shading\(^4\). A BSDF is described by a function of one of the following forms.

\[
\begin{align*}
    f(x, \omega_i, \omega_o) \\
    f(x, \Psi \rightarrow \Theta) \\
    f(x, \theta_i, \phi_i, \theta_o, \phi_o)
\end{align*}
\]  

All three of these forms describe the same idea: at a given point \(x\), the BSDF gives the fraction of radiance coming in from one direction (denoted in spherical coordinates as \(\omega_i, \Psi\), or \(\theta_i, \phi_i\)) leaving that same point in another direction (denoted as \(\omega_o, \Theta\), or \(\theta_o, \phi_o\)). This paper will generally use the first or second forms of the BSDF for brevity.

Given one of these functions and a pair of rays (incoming and outgoing), the entire local illumination of the surface can be defined. The BSDF considers the distribution of radiance over the entire unit sphere, but many surfaces can be described by a simpler BRDF or BTDF (reflectance and transmittance, respectively) function defined over the unit hemisphere on the surface in question. One simple example of a hemispherical BSDF is the lambertian BRDF, a surface that scatters incoming radiance equally in all directions, but only from the same side of the surface as the incoming light. The appearance of paper is a reasonable approximation of this behavior.

As a distribution function, the BSDF must satisfy the normalization condition that it integrate to no more than one for all incoming directions over the unit sphere, or a hemisphere in case of a BRDF or BTDF. More simply, this condition is a statement of conservation of energy; a physically-valid BSDF cannot scatter more light than it takes in. A naive lambertian function \(f(x, \omega_i, \omega_o) = \rho\), where \(\rho \in [0, 1]\) is a constant "reflectivity," is incorrect because it violates this constraint. This can be shown by integrating\(^4\) However, the BSDF cannot capture subsurface scattering effects, such as the translucency of marble or skin.
it over the upper hemisphere $S^+$ with respect to projected outgoing solid angle$^5$:

\[
\int_{S^+} f(\mathbf{x}, \omega_i, \omega_o) \, d\sigma^\perp(\omega_o) = \int_{S^+} \rho \, d\sigma^\perp(\omega_o) \\
= \int_{S^+} \rho |\mathbf{N} \cdot \omega_o| \, d\sigma(\omega_o) \\
= \rho \int_0^{2\pi} \int_0^{\pi/2} \cos(\theta) \sin(\theta) \, d\theta \, d\phi \\
= \rho \int_0^{2\pi} \left( \frac{\sin^2(\theta/2)}{2} \right) \Big|_{\theta=0}^{\pi/4} \, d\phi = \rho \int_0^{2\pi} \frac{1}{2} \, d\phi \\
= \frac{\rho}{2}(2\pi) = \rho \pi
\]

Since the range of this result is $[0, \pi]$, it has to be scaled by $1/\pi$ to make the lambertian a valid BSDF:

\[
f(\mathbf{x}, \omega_i, \omega_o) = \frac{\rho}{\pi}
\]

(2.8)

Since any physically-valid BSDF integrates to 1, it can be used as a distribution to guide importance sampling during ray/path tracing. This can greatly improve the performance of a rendering algorithm by directing samples to where they can gather more information, and will be described later.

### 2.3 The Rendering Equation

In the absence of participating media (which will be assumed from now on$^6$), global illumination can be calculated by the rendering equation, originally described in [Kaj86]. It doesn’t take into account quantum (e.g. diffraction) or temporal (e.g. fluorescence) effects of light due to their complexity, but for very many rendering applications, this approximation is sufficient and produces very realistic results. The basic function of the rendering equation is to produce the outgoing radiance at any given point in a scene

---

$^5$Differential projected solid angle about a given direction $\omega$ is denoted $d\sigma^\perp(\omega)$ and is related to solid angle by the equation

\[
d\sigma^\perp(\omega) = \cos(\theta) \, d\sigma(\omega),
\]

(2.7)

where $\theta$ is angle between $\omega$ and the surface normal $\mathbf{N}$. This evaluation also uses the definition of differential solid angle in spherical coordinates $d\sigma(\omega) = \sin(\theta) \, d\theta \, d\phi$.

$^6$In participating media, the rendering equation has to be modified for attenuation and scattering, though this is will not be discussed in detail in this paper. For more information on the subject, see [LW96].
based on the material description and incoming radiance at that point. This equation can be formulated in several different ways, depending on the domain of integration and representation of direction.

The hemispherical formulation\(^7\) describes radiance output in a specific direction \(L_o(x, \omega_o)\) as a function of incoming radiance \(L_i(x, \omega_i)\) integrated over the hemisphere at a point \(x\):

\[
L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{S^+} L_i(x, \omega_i)f(x, \omega_i, \omega_o)|N_x \cdot \omega_i|d\sigma(\omega_i) \quad (2.9)
\]

In this formulation, \(f\) is the BRDF at the point being considered, \(L_e\) is the emitted radiance from \(x\) in direction \(\omega_o\), and \(N_x\) is the surface normal at \(x\). If using a BSDF, the domain of integration becomes the entire unit sphere rather than a hemisphere.

The area formulation describes radiance output in a specific direction from an infinitesimal surface patch at point \(x\) given radiance output towards \(x\) from surface patches at every point \(y\) in the scene:

\[
L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{M} L_o(y, -\omega_i)f(x, \omega_i, \omega_o)V(x, y)G(x, y)dA_y \quad (2.10)
\]

\(^7\)In favor of Kajiya’s notation, the following formulations are written in a more clear notation similar to that of [Vea98] which will be used in the rest of this paper.
where \( V(x, y) \) is the visibility function (equal to 1 if there is an unobstructed line between \( x \) and \( y \), 0 otherwise), and \( G(x, y) \) is the geometry term:

\[
G(x, y) = \frac{|N_x \cdot \omega| |N_y \cdot -\omega|}{r^2_{xy}} \tag{2.11}
\]

The geometry term captures the foreshortening effect when the two patches are not parallel to each other, and the decreasing solid angle subtended as the patches become further apart.

Also useful is the three-point area formulation of the rendering equation, which is functionally identical to the previous two forms, but uses three explicit surface points in the scene instead of two points and an output direction:

\[
L(y \rightarrow x) = L_e(y \rightarrow x) + \int_M L(z \rightarrow y)f(z \rightarrow y \rightarrow x)V(y \leftrightarrow z)G(y \leftrightarrow z)dA(z) \tag{2.12}
\]

This form describes the contribution to the lighting at a point based on all of the lighting at another, and captures the idea of indirect lighting very explicitly. See [DBBS06] for a more in-depth discussion and illustrations of the rendering equation.

The hemispherical and area formulations describe the same radiance interaction, and can be shown to be identical using the definition of the differential solid angle of a patch \( dA \) relative to some point at a different position:

\[
d\sigma(\omega) = \frac{dA |N_{dA} \cdot -\omega|}{r^2} \tag{2.13}
\]

where \( \omega \) is the direction towards the patch\(^8\), \( N_{dA} \) is the normal direction at the patch, and \( r \) is the distance from the point to the patch. Substituting this into the hemispherical formulation yields:

\[
L_o(x, \omega_o) = L_e(x, \omega_o) + \int_S L_i(x, \omega_i)f(x, \omega_i, \omega_o)|N_x \cdot \omega_i||N_{dA} \cdot -\omega| \frac{dA}{r^2} \tag{2.14}
\]

While in the hemispherical formulation the points \( y \) are implied by the direction \( \omega_i \), the area formulation samples all scene surfaces. To make this change, the domain of integration needs to be changed to all surfaces in the scene \( M \) and the visibility term included so that only surface patches visible in the hemisphere about \( x \) are considered. Combining the two dot products and squared-distance terms yields the geometry term (equation 2.11), producing the area formulation of the rendering equation above (equation 2.10).

\(^8\)The \( \omega \) is negated to clarify that the dot product is considering the magnitude of the patch’s "tilt" away from being perpendicular to the direction from the point being considered.
There is, however, one significant difference between the two formulations. Because the geometry term in the area formulation has a separation term in the denominator ($r^2$), the rendering equation exhibits unbounded behavior when calculating $G(x, y)$ between two closely-spaced, explicitly-selected points. This creates a noticeable problem in bidirectional rendering algorithms, but there exist techniques such as multiple importance sampling, discussed later, which can be used to compensate for this problem in practice.

Note that both sides of the rendering equation include the radiance term $L$, which means that the equation is effectively recursive. Due to the energy-conserving nature of the BSDF term, for a given scene there exists an equilibrium state of incoming and outgoing radiance at every point. While some algorithms, like radiosity algorithms, attempt to find this equilibrium state at a limited spatial resolution, in practice finding this state exactly is impossible for all but the simplest scenes. As such, path tracing and density estimation methods will instead compute the radiance coming into a "camera" in the scene, maintaining at most an approximation of the lighting behavior at any given point in the scene. Camera-based methods produce accurate images, but the camera cannot be moved in the scene without recalculating the global illumination solution (as in real-time rasterization).

### 2.3.1 Importance

Along with radiance, importance can also be transported in a rendering algorithm. Importance captures the contribution to the lighting of some surface patch $S$ of the light leaving any other surface patch in the scene. Rather than calculating the scene importance for every surface in the scene, importance is more often considered to be "emitted" from sensors in the scene, namely the pixels on the image plane of the camera.

Importance is defined as a measure of the influence of a light source on the emitted flux at a particular set of points $S$, $\Phi(S)$. Like radiance, it can be decomposed into direct and indirect components. The direct component, self-importance, is simply
a description of whether a particular point and direction are in the set $S$:

$$W_e(x \leftarrow \Theta) = W_e(x, \omega_i) = \begin{cases} 1 & \text{if } (x, \Theta) \in S \\ 0 & \text{otherwise} \end{cases}. \quad (2.15)$$

This means that if a light source emitting $L(x, \omega_i)$ is in $S$, it fully contributes, otherwise it has no direct contribution on $\Phi(S)$. The left arrow in equation 2.15 captures the idea that the importance at a point is based on another surface patch. This is somewhat strange in the case of self-importance since the same patch is being considered as source and recipient of importance, but the concept is more clear for indirect importance.

Indirect importance captures the same idea, how much some outgoing radiance contributes to $\Phi(S)$, but does so for points outside the $S$. Suppose there is a point $x$ that has an outgoing radiance $L(x, \omega)$, and that radiance is directed towards another point $y = r(x, \omega)$ in the set $S$. The contribution to the outgoing flux of $S$ is then the radiant exitance at $y$:

$$M(y) = \int_{S^+_y} L(x, \omega) f(y, -\omega, \omega_o) |N_y \cdot \omega_o| d\sigma(\omega_o), \quad (2.16)$$

where $S^+_y$ is the surface hemisphere at point $y$. This means that the importance of $L(x, \omega)$ depends on how much of its scattered light at $y$ is important (in the set $S$ directly, or indirectly important by another bounce), and that importance can be thought of as transported identically to radiance, making the indirect importance:

$$W_{ind}(x, \omega) = \int_{S^+_y} W(y, \omega_i) f(y, -\omega, \omega_o) |N_y \cdot \omega_o| d\sigma(\omega_o). \quad (2.17)$$

Combining equations 2.15 and 2.17 yields the importance-based dual of the rendering equation:

$$W(x, \omega) = W_e(x, \omega) + \int_{S^+_y} W(y, \omega_i) f(y, -\omega, \omega_o) |N_y \cdot \omega_o| d\sigma(\omega_o) \quad (2.18)$$

Equation 2.18 describes the actual behavior of many rendering algorithms since they begin tracing at the camera and go outwards, effectively transporting importance until they arrive at an emitter, at which point the contribution to the image is the product of the importance at that point and the emitted radiance back along the ray direction. Again, see [DBBS06] for more detail on the subject.

---

9$r(x, \omega)$ is the ray-casting function and simply means the first point in the scene hit by a ray fired from $x$ in direction $\omega$. 

Chapter 3

Monte Carlo Integration

Monte Carlo integration is a method of computing the numerical integral of an arbitrary function using randomized sampling. It is extremely versatile regarding the functions it can integrate, but has a major shortcoming: its randomized nature causes it to converge very slowly. As Alan Sokal put it in his 1996 lecture, "Monte Carlo is an extremely bad method; it should be used only when all alternative methods are worse" [Sok96]. Monte Carlo is, however, the dominant approach to photorealistic rendering because the rendering equation integral is impossible to evaluate exactly in all but the simplest scenes.

Sections one and two of this chapter will cover basic Monte Carlo integration, as well as error measurement for Monte Carlo algorithms. Section three will discuss several error-reduction techniques used to improve the behavior of Monte Carlo techniques. Section four will then describe how Monte Carlo integration can be applied to rendering and how its performance in such scenarios can be improved.

3.1 Basic Monte Carlo Estimator

The basic idea of the Monte Carlo method is to take numerous random samples of the function being integrated, and to use an estimator to combine them into an approximation of the actual integral value. This method works because the expected value approaches the actual solution as the number of samples increases.
Given a function $f$ to integrate, the correct final value can be denoted $F$,

$$F = \int_{\Omega} f(x) d\mu(x),$$  

(3.1)

where $\Omega$ is the entire domain of integration [DBBS06], and $\mu$ is the Lebesgue measure\(^1\) of the domain [Vea98]. In its simplest form, Monte Carlo uses the estimator:

$$I = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)},$$  

(3.2)

where $N$ is the number of samples ($\{x_1, x_2, ..., x_N\}$), $f$ is the function being integrated, and $p(x_i)$ is the probability density with which sample $x_i$ was chosen\(^2\). This estimator can be shown to converge on the correct integral of $f$ by calculating its expected value and variance.

$$E[I] = E \left[ \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \right]$$

$$= \frac{1}{N} \sum_{i=1}^{N} E \left[ \frac{f(x_i)}{p(x_i)} \right]$$

$$= \frac{1}{N} \sum_{i=1}^{N} \int \frac{f(x)}{p(x)} p(x) \, dx$$

$$= \frac{1}{N} \sum_{i=1}^{N} F$$

$$= \frac{1}{N} F \sum_{i=1}^{N} = \frac{1}{N} F \times N = F.$$

Because its expected value is exactly $F$, this estimator is called an unbiased estimator.

To find the variance of this estimator, the definition of the variance of an equally-

\(^1\)Think of this as an n-dimensional volume of a set. For $n = 1, 2, 3$, this is the measure of length, area, or volume, respectively.

\(^2\)This equation reduces further in the simplest case of uniform sampling on the domain $[0, 1)$ to

$$I = \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

but this will generally not be the case when dealing rendering.
weighted sum is used (assuming the samples are independent):

\[
Var \left( \frac{1}{N} \sum_{i=1}^{N} f(x) \right) = \frac{1}{N^2} \sum_{i=1}^{N} Var(f(x))
\]

\[= \frac{1}{N^2} \cdot N \cdot Var(f(x))
\]

\[= \frac{1}{N} Var\left( f(x) \right). \tag{3.3}\]

Now treating the estimator as an equally-weighted sum of \(f(x)/p(x)\) samples, the definition of variance yields

\[
Var(I) = \frac{1}{N} Var\left( \frac{f(x)}{p(x)} \right)
\]

\[= \frac{1}{N} E \left[ \left( \frac{f(x)}{p(x)} - E\left[ \frac{f(x)}{p(x)} \right] \right)^2 \right]
\]

\[= \frac{1}{N} \int \left( \frac{f(x)}{p(x)} - F \right)^2 p(x) \, dx. \tag{3.4}\]

Since the integral is constant with respect to the function \(f\) and density \(p\), the variance of the Monte Carlo estimator is inversely proportional to \(N\). This means that the standard deviation only decreases with \(\sqrt{N}\), hence the slow convergence of the Monte Carlo method. This convergence rate is the motivation for numerous variance-reduction techniques used to improve Monte Carlo rendering.

### 3.2 Monte Carlo Error

In the context of rendering, the error in a given image is its difference from a mathematically correct reference image. In practice, this reference image is generated by an algorithm known to be correct, though often this takes an extremely long time. Error in rendered images is typically the result of variance across the samples taken for the Monte Carlo estimator, or through a systematic error in the rendering algorithm (bias).

#### 3.2.1 Bias

An algorithm is "biased" if it introduces a systematic error into the results it produces. In rendering this is often the result of approximations made in the algorithm.
to improve its practical performance. For example, a rendering algorithm can introduce bias by "[assuming] that all indirect illumination has low spatial frequency and therefore [estimates] indirect bounces by interpolating among a small set of samples" [Cra06]. By definition, an estimator $I$ is unbiased if its expected error (the bias) is 0 regardless of the number of samples taken:

$$B[I] = E[I_N - F] = 0,$$

(3.5)

where $I_N$ is an estimator taking $N$ samples and $F$ is the true value of the quantity being estimated (in the case of rendering, the reference image).

A biased algorithm does not necessarily produce an incorrect image however, since in some renderers the error introduced can be bounded by a value that decreases with successive samples\(^3\). This is where consistency comes into play.

### 3.2.2 Consistency

A Monte Carlo algorithm is said to be "consistent" if its approximation error approaches zero as more samples are used. As mentioned above, it is possible for an algorithm to be biased and consistent, therefore guaranteeing that eventually it will produce the correct result. An example of such an algorithm is photon mapping, which uses a photon density estimation for indirect lighting; photon mapping has a decreasing bias as the number of sampled photons increases. Mathematically, consistency is defined as:

$$\lim_{N \to \infty} P[|I_N - F| > \epsilon] = 0,$$

(3.6)

for an arbitrary error value $\epsilon$. For a more in-depth discussion of bias and consistency in the context of rendering, see [Cra06].

### 3.3 Variance Reduction

Monte Carlo algorithms introduce variance due to their randomized sampling pattern. In rendering, variance manifests as visual noise, generally worsening significantly in regions lit by complex phenomena (such as caustics). In particular, the area

\(^3\)Note that this bound may be probabilistic in nature and can depend heavily on the scene in question.
formulation of the rendering equation tends to produce very bright pixels, often called "fireflies," in corners of the geometry due to the separation singularity in the geometry term. There are also methods commonly used in rendering that increase variance as a side-effect to increasing efficiency, a notable example being Russian Roulette, where samples are probabilistically discarded based on some simple criteria to improve performance.

As such, there exist numerous methods for variance reduction in rendering (and Monte Carlo algorithms in general). Examples of these are importance sampling, multiple importance sampling, and stratified sampling, though many others exist as well. For the purposes of this paper, the first two are particularly important since they are the primary tools for dealing with error in bidirectional path tracing.

3.3.1 Importance Sampling

Importance sampling\(^4\) is a technique where the sampling density used for a Monte Carlo integration is changed to more closely approximate the function being integrated, thus reducing variance. When the Monte Carlo method was described above, importance sampling was implicitly included in the estimator in the probability density \(p(x_i)\). While this density can be chosen to be uniform, making educated guesses about the integrand and modelling the probability distribution over it can improve the convergence rate and reduce errors significantly.

The main question when using importance sampling is how to choose the sampling distribution to use. This depends heavily on the function being integrated, and is ideal when it is equal to the result of the integration to within a (specific) constant scale factor. This can be shown by considering the estimator and how it relates to the true value:

\[
F \approx I \pm \sigma[I].
\]  
(3.7)

By making the approximation that the error is a Gaussian, the standard deviation is roughly the error bound for the estimator, and by minimizing it, the result of the integration obviously approaches the correct value. Again using the equation for the standard

\(^4\)See [PTVF07].
deviation of an equally-weighted sum, this approximation can be rewritten as

$$ F \approx I \pm \sqrt{\frac{1}{N} \left( E \left[ \frac{f(x)^2}{p(x)^2} \right] - E \left[ \frac{f(x)}{p(x)} \right]^2 \right) } \quad (3.8) $$

The error term above is minimized when the numerator under the square root is minimized. It can be shown that the minimizing probability density is

$$ p(x) = \frac{|f(x)|}{\int_{\Omega} |f(x)| \, dx} \quad (3.9) $$

The denominator in this probability density is a requirement for it to integrate to 1 over the entire domain, but it is exactly the result of the integration being computed, making this density impossible to compute without already having the final result.

In practice, the chosen probability density should roughly model the function being integrated while still being easily computable. This way, $f(x)/p(x)$ approaches a constant and the standard deviation is reduced. A common example in rendering is using cosine-weighted hemispherical sampling. Since the rendering equation has a cosine term, by sampling with respect to this cosine, the rendering variance can be significantly and cheaply reduced. More complex importance sampling can use other terms in the rendering equation, a common choice being the BSDF of the surface in question.

### 3.3.2 Multiple Importance Sampling

When integrating a more complex function, it can be easier to approximate it with different functions over different parts of its domain. For example, in one dimension, a function with two peaks can be approximated with two different Gaussian distributions. With importance sampling, only one function can be used to reduce variance. However, in his 1995 paper, Eric Veach described multiple importance sampling, a method of combining samples from multiple probability density distributions [VG95].

Multiple Importance Sampling gathers samples from multiple probability distributions, referred to as $p_i$, with a given fraction $c_i$ of the total number of samples gathered from each distribution. In this framework, multiple estimators are used and their samples are weighted depending on which probability distribution $p_i$ they are drawn from. Calling the weighting functions $\{w_1, ..., w_s\}$, corresponding to the probability distributions $\{p_1, ..., p_s\}$, the following combined estimator implements multiple importance
sampling:

\[ I = \sum_{i=1}^{s} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(x_{ij}) \frac{f(x_{ij})}{p_i(x_{ij})}, \quad (3.10) \]

where the \( x_{ij} \) are the independent \( j \)-th samples gathered from their respective distributions \( p_i \). This estimator can be made unbiased by upholding the constraints\(^5\)

\[ \sum_{i=1}^{s} w_i(x) = 1 \quad \forall x, \quad (3.11) \]

and

\[ w_i(x) = 0 \quad \text{whenever} \quad p_i(x) = 0. \quad (3.12) \]

These constraints can easily be shown to make the estimator unbiased:

\[
E[I] = E \left[ \sum_{i=1}^{s} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(x_{ij}) \frac{f(x_{ij})}{p_i(x_{ij})} \right] \\
= \sum_{i=1}^{s} \frac{1}{n_i} \sum_{j=1}^{n_i} \int w_i(x) \frac{f(x)}{p_i(x)} p_i(x) \, dx \\
= \sum_{i=1}^{s} \frac{1}{n_i} \sum_{j=1}^{n_i} \int w_i(x) f(x) \, dx \\
= \int \left( \sum_{i=1}^{s} w_i(x) \right) f(x) \, dx \\
= \int f(x) \, dx \\
= F.
\]

While this estimator is unbiased, its convergence rate depends greatly on the choice of weights for each sample. Veach describes a number of heuristics in his paper. The balance heuristic removes the sample value dependence\(^6\) on \( i \), and is defined as

\[ w_i(x_i) = \frac{c_i p_i(x)}{\sum_{j=1}^{s} c_j p_j(x)}. \quad (3.13) \]

Veach proves that the balance heuristic is good by defining an "improvement bound" for arbitrary weighting strategies (his Theorem 1):

\[ Var(\bar{I}) \leq Var(I) + \left( \frac{1}{\min_i n_i} - \frac{1}{\sum_{i=1}^{s} n_i} \right) F^2, \quad (3.14) \]

\(^5\)See [Vea98, p.260].
\(^6\)The \( p_i \) terms cancel.
where \( \bar{I} \) is an estimator using the balance heuristic, and \( I \) is an estimator using arbitrary weights (obeying the weight constraint above). This means that there is a limit on how much a weighting scheme can improve on the variance of the balance heuristic, and that limit goes to 0 as \( N \) increases, assuming all the \( n_i \) are increased.

Two other heuristic Veach defines are the cutoff heuristic \([VG95]\),

\[
w_i = \begin{cases} 
0, & \text{if } p_i < \alpha p_{\text{max}} \\
\frac{p_i}{\sum_{j=1}^{s} \{p_j \mid p_j \geq \alpha p_{\text{max}}\}}, & \text{otherwise},
\end{cases}
\]

(3.15)

where \( p_{\text{max}} = \max_j p_j \), and the power heuristic,

\[
w_i = \frac{p_i^\beta}{\sum_{j=1}^{s} p_j^\beta},
\]

(3.16)

both of which reduce to the balance heuristic if \( \alpha = 0 \) or \( \beta = 1 \). All of these heuristics conform to the weighting constraint (equation 3.11) and have the useful property that the individual probabilities only matter up to a scale factor, which makes their implementation in bidirectional path tracing significantly simpler, as will be shown later.

### 3.4 Monte Carlo Rendering

The rendering equation, as it was described in chapter 2, explicitly describes light interaction at a surface, and to use it to solve the global illumination problem, it must be applied to every surface relevant to the output, possibly including every point in the scene. This is obviously an impossible task for all but the simplest scenes, but since the rendering equation is an integral equation, Monte Carlo integration is very well suited to evaluating it. For now, direct application of the rendering equation will be considered, though later a different formulation will be described for use in bidirectional path tracing.

To calculate the illumination at a given point in the scene, evaluation of the rendering equation can be decomposed into two separate operations: tracing and shading. Tracing is simply transferring radiance from one point in a scene to another along a ray:

\[
L_o(x, \omega) = L_i(r(x, \omega), -\omega),
\]

(3.17)
where $r(x, \omega)$ is the ray-casting function seen earlier. This equation implements the transfer of radiance along a ray from $x$ in direction $\omega$, and reflects the conservation of radiance along that ray. The other component of the rendering equation, shading, should look familiar:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{S^+} L_i(x, \omega_i)f(x, \omega_i, \omega_o)|N_x \cdot \omega_i|d\sigma(\omega_i). \quad (3.18)$$

This is the same rendering equation described earlier, but combined with equation 3.17, it can become fully recursive, considering only outgoing or incoming radiance:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{S^+} L_i(r(x, \omega_i), -\omega_i)f(x, \omega_i, \omega_o)|N_x \cdot \omega_i|d\sigma(\omega_i), \quad \text{or} \quad (3.19)$$

$$L_i(x, \omega_i) = L_e(r(x, \omega_i), -\omega_i) + \int_{S^+} L_i(r(x, \omega_i), \omega'_i)f(x, -\omega_i, \omega'_i)|N_{r(x,\omega_i)} \cdot \omega'_i|d\sigma(\omega'_i). \quad (3.20)$$

The incoming form of the recursive rendering equation is most directly applicable to simple ray tracing since it can begin with incoming radiance at the camera and work outwards. In both distribution ray tracing and path tracing, the integral in this equation is evaluated using a Monte Carlo integration at every bounce, the only difference being how many samples are taken (many and one, respectively). The details of these algorithms will be covered more concretely in the next chapter.

### 3.4.1 Performance Improvements

In equations 3.20 and 3.19, there is no explicit limiting factor on how many times to recurse, or what number of samples (secondary rays) to take for each integration. While these values can be fixed, setting them to a specific constant can introduce bias or harm performance. Instead, there are two closely-related probabilistic approaches to deciding the recursion depth and number of secondary rays.

**Russian Roulette**

Russian roulette is a technique for reducing the number of samples taken by an estimator by probabilistically choosing to not take potentially-irrelevant samples. This approach is generally useful for estimators that involve taking a sum across multiple
random variables. In rendering, these random variables are the rays or paths traced through the scene.

Suppose there exists an estimator $F$ defined as a sum of $N$ estimators $F_i$:

$$F = F_1 + F_1 + \cdots + F_N \quad (3.21)$$

In this estimator, each $F_i$ can be replaced with a Russian roulette estimator $F_i'$:

$$F_i' = \begin{cases} \frac{1}{q_i} F_i, & \text{with probability } q_i \\ 0, & \text{otherwise} \end{cases} \quad (3.22)$$

where $q_i$ is some probability for actually taking the sample(s) required for $F_i$ [Vea98, p.67]. This probability can be set arbitrarily, but it is generally best to choose the value based on some easily evaluated heuristic for the potential contribution of $F_i$. If $F_i$ is unbiased, regardless of what $q_i$ is chosen, $F_i'$ is also unbiased:

$$E[F_i'] = q_i \cdot \frac{1}{q_i} E[F_i] + (1 - q_i) \cdot 0 = E[F_i] \quad (3.23)$$

Due to the addition of another source of randomness to the Monte Carlo integration, Russian roulette obviously increases variance, but in rendering, this comes with the benefit of significantly increasing the algorithm’s efficiency. Specifically, it allows a rendering algorithm to remain unbiased while still being able to skip evaluations unlikely to contribute much to the end result.

Splitting

Splitting is a closely-related technique to Russian roulette, only differing by choosing to take additional estimates rather than discarding them. In this case, the new estimator $F_i'$ is defined

$$F_i' = F_i = \frac{1}{k} \sum_{j=1}^{k} F_{i,j} \quad (3.24)$$

where $k$ is some probabilistically-chosen "splitting factor" and $F_{i,j}$ are individual samples taken independently [Vea98, p.68]. Again, this factor is best decided during rendering based on potential contribution\(^7\).

\(^7\)Note that when probabilistically choosing $k$, the actual estimator will be

$$F_i' = \frac{1}{p_k} \frac{1}{k} \sum_{j=1}^{k} F_{i,j}$$
Note that Russian roulette and splitting can be combined into a single estimator by incorporating a 0-sample case and choosing from two probability distributions (one for the discard probability, \( p_i \), and one for the splitting factor selection, \( q_{i,k} \)):

\[
F'_i = \begin{cases} 
\frac{1}{p_i} F_i, & \text{with probability } q_{i,1} p_i \\
\frac{1}{p_i} \frac{1}{2} (F_{i,1} + F_{i,2}), & \text{with probability } q_{i,2} p_i \\
\frac{1}{p_i} \frac{1}{3} (F_{i,1} + F_{i,2} + F_{i,3}), & \text{with probability } q_{i,3} p_i \\
\ldots & \\
0 & \text{otherwise.}
\end{cases}
\] (3.25)

Using this formulation requires only the constraint that \( \sum_k q_{i,k} = 1 \), and can be shown to have the same expected value as the underlying sampling method:

\[
E[F'_i] = q_{i,1} p_i \left( \frac{1}{p_i} E[F_i] \right) + q_{i,2} p_i \left( \frac{1}{p_i} \left( E[F_{i,1}] + E[F_{i,2}] \right) \right) + \ldots + (1 - p_i) \cdot 0 \\
= \sum_k q_{i,k} \frac{1}{k} E[F_i] \\
= E[F_i] \sum_k q_{i,k} \\
= E[F_i].
\]

where \( p_k \) is the probability of choosing that particular \( k \) value. It is easy to show that this does not create bias by finding the expected value.
Chapter 4

Rendering Algorithms

Turner Whitted originally described the idea of ray tracing in 1979 [Whi80], his major advancement being the inclusion of explicitly-traced secondary rays, allowing the rendering of fairly-accurate reflective and transmissive materials. However, Whitted’s algorithm did not accurately model global illumination for several reasons, foremost of which was the inability to model "soft" phenomena like soft shadows and depth of field. Whitted’s algorithm generated secondary rays exclusively via reflection across surface normals and refraction with Snell’s law, which cannot model "soft" phenomena since surfaces reflect light based on all incoming light, not just the light along the specular reflection direction.

In 1984, Cook, Porter, and Carpenter attempted to solve this problem with an algorithm called "distributed ray tracing" [CPC84]. This algorithm added random sampling to Whitted ray tracing, allowing depth of field, motion blur, soft shadows, and diffuse reflection, among other effects. However, distributed ray tracing was not a true global illumination solution since all of these effects were created via special cases.

It wasn’t until the rendering equation was introduced in 1986\(^1\) that truly realistic rendering could be achieved. The rendering equation allowed for the creation of the path tracing algorithm. Basic path tracing is very similar to distributed ray tracing, but

\(^1\)For example, to create soft shadows, distributed ray tracing explicitly takes multiple samples across area lights. These are not invalid techniques and are commonly used as optimizations in modern algorithms, but they are not grounded in a single, cohesive mathematical framework, and cannot be globally applied.

\(^2\)The rendering equation was described simultaneously by James Kajiya [Kaj86], and Immel et al. [ICG86], though is more often attributed to the former.
instead of special cases for various phenomena, the use of multiple samples for a Monte Carlo estimator allows for surface descriptions to describe all lighting effects in a scene.

Path tracing is capable of solving the global illumination problem and producing extremely realistic images. Naturally, this comes at the cost of increased computational requirements, though many algorithms have been built upon basic path tracing that greatly improve rendering efficiency. This paper will focus mainly on Bidirectional Path Tracing, though other methods such as Metropolis Light Transport [VG97], Real-Time Path Tracing, and Ray Tracing with Irradiance Caching [WRC88] also exist.

4.1 Path Tracers

4.1.1 Basic Path Tracing

Basic path tracing functions by repeatedly evaluating the rendering equation at a set of connected points (the path) in a scene. Usually, this is done by tracing a ray until it hits a surface, using the surface characteristics to choose a reflection direction (assuming the surface doesn’t absorb light completely), and evaluating the rendering equation at that point using a single-sample Monte Carlo estimation. Since the rendering equation is recursive, a new ray is then generated in the reflection direction and the process is repeated.

For the purposes of this paper, a path is defined as an ordered set of points in the scene (vertices) connected with edges, and a path of length $k$ has the form:

$$\bar{x} = x_0x_1 \ldots x_{k-1}x_k.$$  \hfill (4.1)

There are two forms of path tracing, depending on where the paths begin. In the "gathering" form, the paths start at the camera, as per traditional ray tracing, and in the "shooting" form, generally called light tracing, the paths begin at light sources. These forms can produce very different results depending on the scene, and what the user chooses to record as the result.

---

3While this is strictly true, in practice, path tracing would take an unreasonably large amount of time/computation to solve for certain visual effects (such as caustics, for example).

4Shooting and gathering are general terms that refer to the direction of evaluation and apply to more than just path tracing. For example, photon mapping uses both during its execution.
Gathering

Gathering is named as such since it begins at the camera and explores the scene outward, "gathering" any light that would hit the camera sensor. This is mathematically expressed by the measurement equation, which computes a measurement $I_j$ for the $j$th image element being recorded (pixels in this case)[Vea98]:

$$I_j = \int_{M \times S^2} W_r^{(j)}(x, \omega) L_i(x, \omega) dA(x) d\sigma_x^\perp(\omega). \quad (4.2)$$

Note that this equation uses self-emitted importance, and that there is an individual importance function $W_r^{(j)}(x, \omega)$ for each sensor $j$, so while the integration is over all surfaces in the scene, it only has nonzero contributions at sensor surfaces. What this equation represents is capturing the total amount of incoming radiance across a sensor's surface. There also exist other forms of the measurement equation depending on which quantity, radiance or importance, is transported, and in which direction. See [DBBS06] for more details on this subject.

In practice, gathering path tracing uses a Monte Carlo evaluation of the measurement equation for each pixel:

$$I_j = \sum_{k=1}^{N} \frac{L_i(x_k, \omega_k) \cos(N_j, \omega_k)}{p_j(x_k, \omega_k)} \quad (4.3)$$

where $(x_k, \omega_k)$ is the initial ray in the path, and the $N$ initial position samples $x_k$ are taken only from the surface of sensor $j$. $N_j$ is the normal direction at sensor element $j$. Sampling only from sensor surfaces (according to some probability distribution $p_j$) captures the self-emitted importance term implicitly. Obviously, evaluating this estimator requires finding $L_i$ for points and incoming directions on the sensor, and the rendering equation provides a means of calculating this, again using a Monte Carlo estimator.

To do this, consider equation 3.19:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{S^+} L_o(r(x, \omega_i), -\omega_i) f(x, \omega_i, \omega_o) |N_x \cdot \omega_i| d\sigma(\omega_i).$$

Clearly, this equation is recursive in nature, and path tracing consists mostly of evaluating this function some number of times using Monte Carlo methods. Once the initial

---

5With a foreshortening term coming from the projected solid angle in the integration.
"camera ray" generated by the sampling of the measurement equation hits a surface, another Monte Carlo estimator is used with a single sample to find the amount of radiance travelling back along that ray:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \frac{L_o(r(x, \omega_i), -\omega_i) f(x, \omega_i, \omega_o) |N_x \cdot \omega_i|}{p(x \rightarrow r(x, \omega_i))},$$ (4.4)

This estimator returns the amount of emitted radiance at the hit point plus the radiance reflected at the point from some other direction. This direction is randomly sampled, and generates a secondary ray which can hit another point in the scene, hence the recursive nature of the algorithm\(^6\).

In effect, the value returned by the tracing to the measurement equation estimator is

$$L_d(x, \omega) = L_e(x', -\omega) + \frac{f(x', -\omega, \omega') |N_{x'} \cdot \omega'|}{p(x' \rightarrow x'')} \left( L_e(x'', -\omega'') + \frac{f(x'', -\omega'', \omega'') |N_{x''} \cdot \omega''|}{p(x'' \rightarrow x''')} \left( L_e(x''', -\omega''') + \cdots \right) \right),$$ (4.5)

where successive bounces are added into a series with each term multiplied by a BSDF and a foreshortening term, and divided by the probability density (with respect to solid angle) that the next point is generated. Figure 4.1 illustrates the involved vertices and directions.

---

\(^6\)Note that this recursion will not stop until the secondary ray hits a totally-absorbing material, it leaves the scene, or the algorithm arbitrarily stops. The third option is exactly where Russian roulette becomes useful for path tracing.
By taking many initial samples for the unbiased measurement equation estima-
tor, eventually the result will converge to a correct measurement for the pixel without
explicitly solving for the scene’s global illumination. Unfortunately, with basic path
tracing, the amount of time required to render complex lighting effects completely is
effectively infinite (most easily seen with caustics, as in figure 4.2). This problem is
compounded by the fact that each estimator evaluation will only get a contribution if
the path happens to hit an emissive surface, which for many scenes is a low-probability
event (and completely impossible for point lights).

**Shooting**

The shooting form of path tracing, also referred to as particle or light tracing,
reverses the direction of the traced paths, starting them at sampled positions on emissive
surfaces. The measurement equation remains unchanged, and paths will only contribute
if they hit a sensor. Likewise, the rendering equation is also used at each bounce, sam-
Figure 4.3: Path tracing next event estimation: explicit direct lighting contributions at each bounce in the traced path.

pling ω₀ instead of ωᵢ:

\[ L_0(x, \omega_0) = L_e(x, \omega_0) + \frac{L_o(r(x, \omega_i), -\omega_i)f(x, \omega_i, \omega_o)|N_x \cdot \omega_i|}{p(\omega_o)} \quad (4.6) \]

Used on its own, light tracing is extremely useful when precalculating scene lighting for realtime rendering, as each surface in the scene can be considered a sensor. For rendering a single image from a camera, however, particle tracing is almost always extremely inefficient since many paths will not hit the camera’s sensor.

**Next Event Estimation**

Next event estimation is a simple optimization that can be used for both forms of path tracing. It greatly improves their efficiency by taking advantage of the fact that incoming radiance can be split into direct and indirect lighting, which can be considered individually. Used with path tracing, this technique explicitly lights each vertex in the traced path with a sampled point on a light source\(^7\), as shown in figure 4.3. This decreases variance significantly since each path has a much higher chance of providing contributions to the measurement equation estimator.

A point is directly lit by a given light source if there is an unobstructed line connecting them, meaning it is lit without any surface interaction. The point is indirectly lit if the light reaching it has interacted (reflection or transmission) with at least one

\(^7\)Used with light tracing, each path vertex is explicitly "measured" from a point sampled on a sensor. For simplicity, this approach will not be discussed in detail here.
surface along the way, forming a light path. At any given point, the decomposition into these components is trivial:

\[ L(x, \omega) = L_{\text{direct}}(x, \omega) + L_{\text{indirect}}(x, \omega). \] (4.7)

The benefit of this decomposition is that, while the chances of randomly hitting an emissive object by a path may be small, if at each bounce the direct component is found by explicitly connecting to an emissive surface, the direct component will very often be non-zero, giving the path an actual contribution. However, there are two issues that arise with this technique: how to compute the direct contribution, and how to handle the path actually intersecting an emissive surface.

For the direct contribution, the hemispherical rendering equation (2.9) is difficult to use since its estimator would require the probability of the direction to the light source being sampled, while the actual sampling is not performed this way. Instead, the area formulation of the rendering equation (2.10) is more applicable since its domain of integration is over scene surfaces, thus matching the sampling method for next event estimation. The estimator for the direct contribution is then

\[ L_{\text{direct}}(x, \omega_o) \approx \frac{L_e(y, -\omega_i) f(x, \omega_i, \omega_o) V(x, y) G(x, y)}{p(y)}, \] (4.8)

where \( V(x, y) \) and \( G(x, y) \) are the visibility and geometry (equation 2.11) terms defined earlier. The \( p(y) \) term is the probability density with respect to area that point \( y \) on the light was chosen.

If the path hits an emissive surface during tracing, normally the emitted radiance would be added into the running sum for the path. However, this contribution is direct lighting for the previous vertex in the path, which in most cases will create a "double counting" effect that incorrectly biases the estimator. To overcome this, the emissive component \( L_e \) in equation 4.4 needs to be either ignored or combined with the direct lighting estimate for the previous vertex via weighted average\(^8\). The indirect contribution is simply whatever results from continued tracing in the algorithm, as before.

\(^8\)If combining, the BSDF, foreshortening term, and probability density terms need to also be taken into account. This is best done using multiple importance sampling.
4.2 Bidirectional Path Tracing

Both forms of path tracing suffer from problems due to being unidirectional. Gathering misses small or obscured light sources, and shooting may take samples of irrelevant scene regions (missing sensors, effectively). The symmetry of these issues, namely that both can undersample useful regions of the scene, leads naturally to the idea of using both methods simultaneously.

Using both gathering and shooting paths is precisely what bidirectional path tracing (BDPT) does. Introduced first by Lafortune and Willems [LW93] and refined by Veach and Guibas [VG95], bidirectional path tracing traces paths simultaneously from lights and sensors, then connects them to create complete paths. These complete paths, if their connecting edges are unobstructed, are likely to contribute to the final rendering result, thus making BDPT very efficient.

Both the Veach and LaFortune algorithms have been implemented many times and shown to be effective. However, this thesis will focus primarily on the Veach formulation of bidirectional path tracing due to its greater mathematical rigor and complete description of path weighting schemes.

4.2.1 Path Integral Formulation

The basis of Veach’s bidirectional path tracing algorithm is a reformulation of the measurement equation. Instead of integrating over the surfaces of all sensors, BDPT uses an integration over the space of all possible paths in the scene of the form

\[ I_j = \int_{\Omega} f_j(\bar{x})d\mu(\bar{x}). \]  

(4.9)

The path space \( \Omega \) is the set of transport paths of all finite lengths, defined as

\[ \Omega = \bigcup_{k=1}^{\infty} \Omega_k, \]  

(4.10)

where \( \Omega_k \) is the set of paths of length \( k \) [Vea98, p. 222]. A path of length \( k \) is a sequence of \( k + 1 \) surface points (vertices) in the scene,

\[ \bar{x} = x_0x_1\ldots x_k. \]  

(4.11)

\(^9\)Detailed very thoroughly in his Ph.D. dissertation [Vea98].
and the integration over path space is taken with respect to \( \mu \), the area-product measure on paths, defined for a set of paths \( D \subset \Omega_k \) as
\[
\mu_k(D) = \int_D dA(x_0) \ldots dA(x_k).
\] (4.12)

This measure can be extended to any set of paths \( D \subset \Omega \) by separating the set by length and finding the sum of the measures of each part:
\[
\mu(D) = \sum_{k=1}^{\infty} \mu_k(D \cap \Omega_k).
\] (4.13)

This measure corresponds to using the area formulation of the rendering equation (2.10) and can be used to derive the path integral measurement equation (4.9) from the standard measurement equation (4.2).

First, we define the relation between the projected solid angle measure at a point \( y \) and the area measure at a point \( x \),
\[
d\sigma^\perp_y(\omega_i) = |N_y \cdot \omega_i||N_x \cdot -\omega_i| \frac{dA(x)}{|x - y|^2} = G(x \leftrightarrow y)dA(x).
\] (4.14)

Using this, the measurement equation can be redefined in terms of area measures\(^{10}\),
\[
I_j = \int_{M \times S^2} W_e^{(j)}(x, \omega)L_e(x, \omega)dA(x)d\sigma^\perp_x(\omega)
\]
\[
= \int_{M \times M} W_e^{(j)}(x \leftarrow y)L(x \leftarrow y)G(x \leftrightarrow y)dA(x)dA(y).
\] (4.15)

This form allows the measurement equation to be recursively expanded using the area formulation of the rendering equation:
\[
I_j = \sum_{k=1}^{\infty} \int_{M^{k+1}} L_e(x_0 \rightarrow x_1)G(x_0 \leftrightarrow x_1)
\]
\[
\cdot \prod_{i=1}^{k-1} f_s(x_{i-1} \rightarrow x_i \rightarrow x_{i+1})G(x_i \leftrightarrow x_{i+1})
\]
\[
\cdot W_e^{(j)}(x_{k-1} \rightarrow x_k)dA(x_0) \ldots dA(x_k).
\] (4.16)

The products are included to highlight the three parts of this expansion since this form of the measurement equation directly represents rays of light bouncing through a scene.

\(^{10}\)This is essentially a reproduction of the derivation in [Vea98, p.221].
The three parts of the integrand in equation 4.16 are, respectively, radiance emitted from a light source, reflections at \( k - 1 \) surfaces in the scene, and the importance of radiance arriving at sensor \( j \). The geometry terms capture all foreshortening effects between each vertex in the path, effectively linking them together. In this form, the integral becomes an integration over all paths of length \( k \), and the area-product measure naturally comes out of the expansion by combining the differential area terms \( dA(x_0) \cdots dA(x_k) \). Finally, the summation on the left captures that this measurement is taken over paths of all lengths.

Now, the area formulation measurement equation can be found by defining the measurement contribution function \( f_j(\bar{x}) \) as the integrand of equation 4.16 with \( k \) equal to the length of \( \bar{x} \). Defining \( f_j \) this way allows the explicit integration over all of path space, yielding the new path integral measurement equation 4.9.

\[
f_j(\bar{x}) = L_e(x_0 \rightarrow x_1)G(x_0 \leftrightarrow x_1) \prod_{i=1}^{k-1} f_s(x_{i-1} \rightarrow x_i \rightarrow x_{i+1})G(x_i \leftrightarrow x_{i+1})W_e^{(i)}(x_{k-1} \rightarrow x_k)
\]

(4.17)

4.2.2 Path Sampling

Using the path integral formulation requires some method of sampling path space, which cannot be directly sampled since it depends on the scene being rendered and is infinite-dimensional\(^{11}\). However, techniques from path tracing can be applied to sample path space based on local information. Taken together, these techniques are called local path sampling.

Local Path Sampling

Local path sampling algorithms, while not the only way of generating samples of path space, are techniques that generate path vertices one at a time based on the information at already-sampled vertices (often the BSDFs at these vertices)[Vea98]. Local path sampling consists of three basic mechanisms:

- **Area Sampling** Points can be sampled directly on a surface based on a density distribution over that surface. For example, this is often the way that points are

\(^{11}\)Path space is infinite-dimensional because a path can contain any number of vertices, and path space contains all paths by definition.
sampled on area lights, with either uniform probability or based on the radiant exitance of the light surface. This technique can be used on any well-defined surface, so is also applicable to finite-aperture camera lenses and general scene surfaces.

- **Hemispherical Sampling** Another method for choosing vertices is by sampling a direction at an existing vertex and tracing a ray from that point to another surface in the scene. This is the typical method used for path tracing, and can be used to easily importance sample surface BSDFs, or simply the cosine foreshortening term in the rendering equation. For lights, this hemispherical sampling can be used to select initial emission directions (for shooting algorithms).

- **Explicit Connection** Two existing vertices can be connected explicitly, creating a new path edge between them, subject to a visibility test. This method involves no probability density since it relies in having already-sampled vertices to connect.

**Path Probability**

Importance-sampled Monte Carlo estimators require the probability density of each sample taken, but local path sampling consists of taking samples from multiple probability density functions. Since the path integral measurement equation uses the area-product measure \( \mu \), each path’s probability density must be measured with respect to that measure. Given a path \( \bar{x} \), this probability density is,

\[
p(\bar{x}) = \frac{dP}{d\mu}(\bar{x}) = \frac{dP}{d\mu}(x_0 \ldots x_k) = \prod_{i=0}^{k} \frac{dP}{dA}(x_i),
\]

as is described in [Vea98, p.227].

This equation can only be applied directly if every path vertex is area-sampled, but doing so is generally difficult, inefficient, or both. Using the relation between differential solid angle and differential area (equation 2.3), hemisphere sampling probability densities can be converted to area sampling densities. Suppose vertex \( y \) is sampled from vertex \( x \) by choosing a direction \( \omega_o \) and tracing a ray from \( x \) in that direction. Given the
probability density of choosing the outgoing direction, \( p(\omega_o) \), the area sampling density can be calculated\(^\text{12}\),

\[
\frac{dP}{dA}(y) = \frac{dP}{d\sigma}(\omega_o) \frac{d\sigma(\omega_o)}{dA(y)} \Rightarrow p(y) = p(\omega_o) \left( \frac{|N_x \cdot \omega_o|}{||y - x||^2} \right). \tag{4.19}
\]

The relation to the probability density with respect to projected solid angle can be found using equation 2.7,

\[
\frac{dP}{d\sigma^\perp}(\omega_o) = \frac{dP}{d\sigma}(\omega_o) \frac{d\sigma(\omega_o)}{d\sigma^\perp(\omega_o)} \Rightarrow p^\perp(\omega_o) = p(\omega_o) \frac{1}{|N_x \cdot \omega_o|}. \tag{4.20}
\]

Combining the two equations above yields the conversion between densities with respect to area and densities with respect to projected solid angle, which is used extensively in bidirectional path tracing,

\[
p(y) = p^\perp(\omega_o) \left( \frac{|N_x \cdot \omega_o||N_y \cdot -\omega_o|}{||y - x||^2} \right) \Rightarrow = p^\perp(\omega_o)G(x, y), \tag{4.21}
\]

where \( G \) is the geometry term (equation 2.11). Using these equations, the probability density of a path generated with hemispherical sampling can be found with respect to the area-product measure.

### 4.2.3 Bidirectional Path Sampling

Bidirectional path tracing consists of generating subpaths from the eye and a light source simultaneously, using area and hemisphere sampling, then explicitly connecting their endpoints to generate a full path. This provides an increased likelihood that the path contributes to the final image since it necessarily connects the light and eye. Different selections of subpath lengths yield different sampling techniques, and given a path length of \( k \), there are \( k + 2 \) techniques possible.

For example, consider a path \( \bar{x} = x_1 \ldots x_4 \) of length \( k = 4 \) where \( x_1 \) is a point on a light source and \( x_4 \) is a point on the camera. This full path can be generated six different

\(^\text{12}\)See [Vea98, p.228].
ways since there are four options for connecting edges, and two more for a light subpath hitting the camera or an eye subpath hitting a light. Using \( s \) to represent the number of light subpath vertices and \( t \) to represent the number of eye subpath vertices, these six different techniques for generating \( \bar{x} \) can be seen in figure 4.4\(^\text{13}\). Each of these techniques has a different probability density \( p_{s,t} \), making them useful since they capture different factors in the measurement contribution function \( f_j \) (equation 4.17, [Vea98, p.298]) and can all be importance sampled.

The bidirectional path tracing algorithm consists of creating samples of path space using all possible techniques, referred to by their probability densities \( p_{s,t} \) from here on, and using a multiple importance sampling estimator to combine them,

\[
F = \sum_{s \geq 0} \sum_{t \geq 0} w_{s,t}(\bar{x}_{s,t}) \frac{f_j(\bar{x}_{s,t})}{p_{s,t}(\bar{x}_{s,t})},
\]

where \( \bar{x}_{s,t} \) is a specific path generated by technique with density \( p_{s,t} \), and \( w_{s,t} \) is the weight assigned to that path by a multiple importance sampling heuristic (such as those in section 3.3.2).

\(^{13}\)Note that technique \( s = 5, t = 0 \) is impossible with a pinhole camera since a random ray cannot hit an infinitesimal point. Likewise, technique \( s = 0, t = 5 \) is impossible for point lights.
Figure 4.5: The four possible prefix-suffix connections of length one subpaths.

Efficient Path Sampling

A naive implementation of bidirectional path tracing creates samples by first randomly selecting the number of vertices in light and eye subpaths \((n_L\) and \(n_E\), respectively), generating subpaths with these lengths, and connecting their endpoints. In practice, this data generated for each subpath can be reused by explicitly connecting the end of every possible prefix of the light path to the end of every possible eye path suffix. Specifically, the algorithm first generates a light subpath

\[ y_0 \ldots y_{n_L-1} \]

and an eye subpath

\[ z_{n_E-1} \ldots z_0, \]

then generates the paths

\[ \bar{x}_{s,t} = y_0 \ldots y_{s-1}z_{t-1} \ldots z_0 = x_0 \ldots x_{s+t-1} \]

with lengths \(k = s + t - 1\), where \(0 \leq s \leq n_L\), \(0 \leq t \leq n_E\), and \(k \geq 1\). Figure 4.5 illustrates these connections for a set of eye and light subpaths with two vertices each \((n_L = n_E = 2)\). This example generates one path of length 1 with \(p_{1,1}\), two of length 2 with \(p_{1,2}\) and \(p_{2,1}\), and one of length 3 with \(p_{2,2}\). While this method or reusing vertices makes the path space samples dependent, in practice this does not affect the results of the algorithm once enough samples are taken (see [VG95, p.9]).

Calculating Contributions

Each generated sample \(\bar{x}_{s,t}\) adds a contribution to the bidirectional path tracing estimator (equation 4.22), which consists of multiple BSDF and geometry terms. Recal-
Calculating these terms for every sample is wasteful, so decomposing these contributions into precomputable terms allows for a much more efficient implementation of the algorithm.

The bidirectional path tracing estimator can be rewritten as a sum of contribution terms,
\[
F = \sum_{s \geq 0} \sum_{t \geq 0} C_{s,t},
\]
where \( C_{s,t} \) is defined as
\[
C_{s,t} \equiv w_{s,t}(\bar{x}_{s,t}) \frac{f_j(\bar{x}_{s,t})}{p_{s,t}(\bar{x}_{s,t})}.
\] (4.23)

This contribution can be further decomposed into the weight term \( w_{s,t} \) and the unweighted contribution \( C^*_{s,t} \). This unweighted contribution can be computed as the product of three terms:
\[
C^*_{s,t} = \frac{f_j(\bar{x}_{s,t})}{p_{s,t}(\bar{x}_{s,t})} = \alpha_L c_{s,t} \alpha^E_t,
\] (4.24)
where \( \alpha_L \) depends only on the light prefix, \( \alpha^E_t \) depends only on the eye prefix, and \( c_{s,t} \) is a connecting term that only depends on the explicit edge connecting the two prefixes \((y_{s-1}, z_{t-1})^\perp\). This decomposition is extremely useful since the \( \alpha \) terms can all be precalculated during the subpath creation stage, so finding each unweighted contribution term requires only the calculation of \( c_{s,t} \) and a simple multiplication.

**Probability Density** The probability density \( p_{s,t} \) for each path \( \bar{x}_{s,t} \) is simply the product of the densities with which the light prefix and eye suffix were generated,
\[
p_{s,t} \equiv p_{s,t}(\bar{x}_{s,t}) = p^L_s p^E_t.
\]

Since the subpaths are generated with local sampling, each vertex has an "accumulated" product of probability densities. For example, suppose vertex \( y_i \) was generated with probability density \( p_{y_i} \), and vertex \( y_{i+1} \) was generated with hemispherical sampling from \( y_i \). Using equations 2.11 and 4.21, the accumulated probability density at \( y_{i+1} \) with respect to area is
\[
p_{y_{i+1}} = P_{\sigma^L}(y_i \rightarrow y_{i+1}) G(y_i \leftrightarrow y_{i+1}) p_{y_i},
\]
\[\text{14See [Vea98, p.302]}\]
where \( P_{\sigma^+}(y_i \rightarrow y_{i+1}) \) is the density with respect to projected solid angle of choosing the direction from \( y_i \) to \( y_{i+1} \), and \( G(y_i \leftrightarrow y_{i+1}) \) is the geometry term between the two points. The same equation applies to points generated on the eye path. Setting the initial values based on direct area sampling densities \( P_A \) yields the definitions of \( p_L^1 \) and \( p_E^1 \), where \( s \) and \( t \) are the number of vertices in the prefix and suffix, respectively:

\[
\begin{align*}
p_L^0 &= 1 \\
p_L^1 &= P_A(y_0) \\
p_L^i &= P_{\sigma^+}(y_{i-2} \rightarrow y_{i-1})G(y_{i-2} \leftrightarrow y_{i-1})p_L^{i-1} & \text{for } i \geq 2 \\
p_E^0 &= 1 \\
p_E^1 &= P_A(z_0) \\
p_E^i &= P_{\sigma^+}(z_{i-2} \rightarrow z_{i-1})G(z_{i-2} \leftrightarrow z_{i-1})p_E^{i-1} & \text{for } i \geq 2.
\end{align*}
\]

(4.25)

**Radiance Decomposition** Precomputing the \( \alpha \) terms requires a decomposition of emitted radiance into spatial and directional components, \( L_e^{(0)} \) and \( L_e^{(1)} \) respectively. The reason for this decomposition is that when there is an explicit connection to a light, the spatial component is known during subpath creation, but the directional component, which depends on the explicit connecting edge, is not. These terms are also analogous to the separate area and hemispherical sampling techniques used with area lights and finite-aperture cameras. They components are defined as

\[
\begin{align*}
L_e^{(0)}(x) &= \int_{S^2} L_e(x, \omega) \ d\sigma^+(\omega) \\
L_e^{(1)}(x, \omega) &= \frac{L_e(x, \omega)}{L_e^{(0)}}.
\end{align*}
\]

(4.26)

The term \( L_e^{(1)}(x, \omega) \) is the directional distribution of radiance over the hemisphere, so it integrates to 1 by definition\(^\text{15} \). Importance is similarly decomposed into \( W_e^{(0)} \) and \( W_e^{(1)} \).

As an example, suppose there exists a lambertian area light with constant emitted radiance \( L_e(x, \omega) = c \). By these definitions, \( L_e^{(0)}(x) = c\pi \) and \( L_e^{(1)}(x, \omega) = \frac{1}{\pi} \). The proof of this is essentially identical to the BSDF validity proof in section 2.2, so it will not be reproduced here.

\(^{15}\text{See [Vea98, p.234].}\)
**Unweighted Contribution** Using the definitions for the path probabilities, similar accumulated $\alpha$ terms can be defined for the light and eye subpaths ($\alpha_L$ and $\alpha_E$, respectively). Each $\alpha$ term contains all of the measurement contribution function (equation 4.17) terms and probability density terms that can be precomputed for the first $i$ vertices of each subpath. This can be thought of as the radiance or importance "throughput" up to but not including vertex $i$ divided by the probability density of having generated vertex $i$.

These $\alpha$ values are defined in a recursive way similar to path probability densities:

\[
\begin{align*}
\alpha^L_0 &= 1 \\
\alpha^L_1 &= \frac{L^{(0)}_e(y_0)}{P_A(y_0)} \\
\alpha^L_i &= \frac{f_s(y_{i-3} \rightarrow y_{i-2} \rightarrow y_{i-1})}{P_{\sigma^+}(y_{i-2} \rightarrow y_{i-1})} \alpha^L_{i-1} \quad \text{for } i \geq 2 \\
\alpha^E_0 &= 1 \\
\alpha^E_1 &= \frac{W^{(0)}_e(z_0)}{P_A(z_0)} \\
\alpha^E_i &= \frac{f_s(z_{i-1} \rightarrow z_{i-2} \rightarrow z_{i-3})}{P_{\sigma^+}(z_{i-2} \rightarrow z_{i-1})} \alpha^E_{i-1} \quad \text{for } i \geq 2.
\end{align*}
\]

The BSDF terms here have a special meaning in that $f_s(y_{i-1} \rightarrow y_0 \rightarrow y_1)$ is defined as $L^{(1)}_e(y_0 \rightarrow y_1)$, and similarly $f_s(z_{i-1} \rightarrow z_0 \rightarrow z_{i-3}) \equiv W^{(1)}_e(z_0 \rightarrow z_i)$. Note that in all of these equations, the geometry terms in the measurement contribution function and the path densities cancel exactly.

The $\alpha$ value for each vertex in both subpaths can be efficiently calculated while tracing the subpaths, and all that is necessary to find the unweighted contribution is a connecting term $c_{s,t}$. This term captures all of the information that cannot be precomputed since it depends on the connection between the light prefix and eye suffix. By going back to the definition of $C^*_{s,t}$ (equation 4.24) and removing the $\alpha$ terms, $c_{s,t}$ is
found to be
\[ c_{0,t} = L_e(z_{t-1} \rightarrow z_{t-2}) \quad \text{(Light hit by eye path),} \]
\[ c_{s,0} = W_e(y_{s-2} \rightarrow y_{s-1}) \quad \text{(Camera hit by light path), and} \]
\[ c_{s,t} = f_s(y_{s-2} \rightarrow y_{s-1} \rightarrow z_{t-1})G(y_{s-1} \rightarrow z_{t-1})f_s(y_{s-1} \rightarrow z_{t-1} \rightarrow z_{t-2}) \quad \text{for } s, t > 0, \]
(4.28)

where the geometry term \( G(y_{s-1} \rightarrow z_{t-1}) \) includes a visibility test. This geometry term creates the primary difficulty in using bidirectional path tracing since it contains an inverse distance factor \((1/r^2)\). If the endpoints of the light prefix and eye suffix are very near each other, the geometry term becomes huge and typically creates a very bright pixel (a "firefly"). This sort of connection, while quite possible, is extremely unlikely, so by using multiple importance sampling, its contribution is reduced proportionately by the weight term \( w_{s,t} \).

### 4.2.4 Multiple Importance Sampling of Paths

The last term that needs to be found is the weighting \( w_{s,t} \equiv w_{x,i}(\bar{x}_{s,t}) \) for each path space sample. This value is found via multiple importance sampling across the \( k + 2 \) techniques capable of generating \( \bar{x}_{s,t} \), and thus requires the probability density with which each technique would have generated the particular sample being weighted. To simplify the notation, let \( p_i \) be the density for generating \( \bar{x}_{s,t} \) using \( i \) light vertices and \( s + t - i \) eye vertices:\(^{16}\)
\[ p_i = p_{i,s+t-i}(\bar{x}_{s,t}) \quad \text{for } i = 0, \ldots, s + t \]
(4.29)

With this definition, \( p_s \) is the probability with which \( \bar{x}_{s,t} \) was actually generated.

To simplify the weighting calculation, the weighting heuristic can be modified to take advantage of the fact that the probability densities only matter up to a scale factor\(^{17}\).

For example, using the power heuristic with \( \beta = 2 \),
\[ w_{s,t} = \frac{p_s^2}{\sum p_i^2} = \frac{1}{\sum_i (p_i/p_s)^2}. \]
(4.30)

\(^{16}\)See [Vea98, p.305].

\(^{17}\)See [Vea98, p.306]
Thus, starting with \( p_i/p_s = 1 \), the ratio \( p_i+1/p_i \) can be repeatedly applied to find the \( p_i/p_s \) terms for \( s + t \geq i > s \), and its reciprocal for \( 0 \leq i < s \).

Going back to the path notation \( \bar{x} = x_0 \ldots x_k \), this ratio is found by noting that the only difference between \( p_i \) and \( p_i+1 \) is how vertex \( x_i \) is generated. At the beginning of the light path, this ratio considers the area sampling density of \( x_0 \) versus the converted hemisphere sampling density of generating \( x_0 \) from \( x_1 \):

\[
\frac{p_1}{p_0} = \frac{P_A(x_0)}{P_{\sigma^+}(x_1 \rightarrow x_0)G(x_1 \leftrightarrow x_0)}. \tag{4.31}
\]

Referring to figure 4.4, this corresponds to the relative probabilities of 4.4a and 4.4c. At the other end of the path, the densities for generating \( x_k \) are similarly related:

\[
\frac{p_{k+1}}{p_k} = \frac{P_{\sigma^+}(x_{k-1} \rightarrow x_k)G(x_{k-1} \leftrightarrow x_k)}{P_A(x_k)}. \tag{4.32}
\]

This corresponds to the relative probabilities of figures 4.4f and 4.4d. Finally, for all the other terms,

\[
\frac{p_{i+1}}{p_i} = \frac{P_{\sigma^+}(x_{i-1} \rightarrow x_i)G(x_{i-1} \leftrightarrow x_i)}{P_{\sigma^+}(x_{i+1} \rightarrow x_i)G(x_{i+1} \leftrightarrow x_i)}, \tag{4.33}
\]

which corresponds to extending the light prefix by one vertex and shortening the eye suffix by one vertex. Using these ratios, all of the terms in the weighting heuristic can be found relative to \( p_i/p_s = 1 \), thus completing the calculation of the weighted contribution:

\[
C_{s,t} = w_{s,t} C_{s,t}^* = w_{s,t} \alpha_{s,t}^L c_{s,t} \alpha_{s,t}^E. \tag{4.34}
\]

### 4.2.5 Special Cases

Bidirectional path tracing is fairly straightforward to implement, given that all of the scene elements, including the lights and camera, are diffuse in nature. Realistic scenes, however, often have transparent and/or reflective specular objects or directional lights, and many renderers have a pinhole camera model. These types of objects have Dirac delta functions in at least one part of their description, and as such, require careful handling for the algorithm to function correctly. Connecting explicitly to the eye also presents a problem in that the contribution from that connection does not necessarily (or even usually) apply to the pixel being sampled at the time, which requires maintaining that data separately. Pinhole cameras present an additional difficulty in that they are not
randomly sampled like a BSDF, requiring the probability density of points sampled from the camera to be calculated differently as well. While this thesis covers the essential special cases for the basic BDPT algorithm, there are additional special case optimizations covered in [Vea98].

**Specular Vertices**

A specular vertex is any point along a sampled path that has at least one property defined with a Dirac delta,

\[
\delta(x) = \begin{cases} 
+\infty & x = 0 \\
0 & x \neq 0 
\end{cases} \quad \text{such that} \quad \int_{-\infty}^{+\infty} \delta(x) dx = 1. \tag{4.35}
\]

Since the Dirac delta is not finite, parts of the path probability or measurement contribution function may not be explicitly computable, or may simply become zero. Some common examples are ideal mirrors, where deltas are elements of the BRDF,

\[
f_s(x, \theta_i, \phi_i, \theta_o, \phi_o) = \rho_s \delta(\theta_i - \theta_o) \delta(\phi_i + \pi + \phi_o), \tag{4.36}
\]

which describes a reflection over the normal at \(x\) with a constant reflectivity term \(\rho_s\), and pinhole cameras, where there is a delta in its sampling density,

\[
p_A(x) = \delta(x - x_{\text{eye}}), \tag{4.37}
\]

which describes an infinitesimal point in space.

Handling specular points in computing unweighted contributions is generally straightforward. Suppose vertex \(x_i\) is an ideal mirror. In that case, if it was generated via sampling from \(x_{i-1}\), sampling vertex \(x_{i+1}\) from \(x_i\) using the BRDF guarantees that \(x_{i+1}\) is along the reflection direction at \(x\). This means that the Dirac delta conditions in equation 4.36 are satisfied by construction, and they can be treated as having a value of 1 in the Monte Carlo integration of the rendering equation\(^{18}\). Specifically, this means

\(^{18}\)More explicitly, the BRDF integrates to 1 and is valid. Since the reflection direction \(\omega_o\) was explicitly generated from the incoming direction \(\omega_i\) by reflecting it over the normal, its probability density contains the same Dirac deltas as the BRDF. These terms cancel, allowing the delta in the contribution to be treated as 1 multiplied by any additional factors (i.e. albedo, color, reflection probability, etc.) if the material defines them.
that computed alpha values are well defined because all of the terms
\[
\frac{f_s(x_i-3 \rightarrow x_i-2 \rightarrow x_{i-1})}{P_\sigma^-(x_{i-2} \rightarrow x_{i-1})}
\]
contain the same Dirac delta in both the numerator and denominator\(^\text{19}\).

However, these Dirac deltas need to be handled carefully when they are involved in an explicit connection, and when calculating path weights. When a specular vertex is explicitly connected to another vertex, the unweighted contribution contains the connecting term
\[
c_{s,t} = f_s(y_{s-2} \rightarrow y_{s-1} \rightarrow z_{t-1})G(y_{s-1} \rightarrow z_{t-1})f_s(y_{s-1} \rightarrow z_{t-1} \rightarrow z_{t-2})
\]
for \(s, t > 0\), where at least one of the BSDF functions \(f_s\) contains a Dirac delta. The delta functions in this term have a zero probability of being nonzero, which would require at least one of the vertices being randomly placed exactly along a reflecting direction from the other. This means that when one of the connecting vertices has a specular component in its BSDF, there is no need to compute that component’s contribution. If a connecting vertex has no nonspecular component, there is no need to perform even a visibility test.

When finding the weight for a given path, even though that path is known to be valid\(^\text{20}\), the probability density conversion (equation 4.33),
\[
\frac{p_{i+1}}{p_i} = \frac{P_\sigma^-(x_{i-1} \rightarrow x_i)G(x_{i-1} \leftrightarrow x_i)}{P_\sigma^+(x_{i+1} \rightarrow x_i)G(x_{i+1} \leftrightarrow x_i)}
\]
can still have non-matching Dirac delta functions in its numerator and denominator\(^\text{21}\). Specifically, this occurs when one of the connecting vertices of a potential path is specular. Since the purpose of this conversion is to find the probability density of generating a particular potential path, even though its vertices line up correctly for a specular surface interaction, the Delta functions would never be satisfied by random sampling, making that potential path’s probability density zero.

In practice, this amounts to labeling every vertex with a specular flag which indicates that the BSDF and probability density of the vertex contain implicit Dirac delta

\(^{19}\)See [Vea98, p.314].

\(^{20}\)This assumes that its unweighted contribution has first been found to be nonzero.

\(^{21}\)See [Vea98, p.315-316] for more detail on handling specular vertices.
functions. As such, the values for these terms should be considered coefficients, capturing any non-delta qualities of the vertex (i.e. reflectivity, reflection versus refraction probability, etc.). When calculating the $p_i$ values for a potential path, if one or both of the connecting vertices is specular, the $p_i$ term is still calculated to keep track of any coefficients involved, but is replaced with zero when used in the MIS heuristic.

**Explicit Eye Connection**

Explicit connection to the eye is extremely important since it is the primary source of low-variance caustic contributions, especially when the scene contains small light sources. The difficulty in this is that when an explicit eye connection is made, the connecting edge can pass through any pixel in the image plane. This contribution can be discarded if it doesn’t pass through the pixel being considered at the time, but it is far more efficient to accumulate all such contributions in a separate "light" image.

The creation of the light image is straightforward. The contribution and weight for the sample are computed as normal, and the resulting weighted contribution is accumulated into the light image pixel corresponding to the image plane pixel intersected by the connecting edge. Note that computing the intersected pixel depends on the camera model being used; finding the pixel is very straightforward for the common pinhole camera, but a simulated lens system may require significant computation.

At the end of tracing, the primary rendering image and the light image need to be combined. The light image contains an accumulation of light energy that isn’t scaled for
the number of samples taken, so the adjustment is made during the image combination.

The images are combined pixel-by-pixel using the following equation\textsuperscript{22},

\[ I_j = \left( |D|/N \right) I_j^L + I_j^E, \]  

(4.38)

where \( N \) is the number of samples and \(|D|\) is size of the image plane.

**Pinhole Camera**

Despite being the most common camera model in rendering, pinhole cameras are infinitesimal in size and can create a number of problems. The specific problem here is that equation 4.38 requires a finite image plane size, but the image plane for a pinhole camera has arbitrary dimensions\textsuperscript{23}. However, by considering a "simulated" image plane of a specific size behind the pinhole, accounting for the projection onto this screen compensates for the \(|D|\) term. For a given explicit eye connection, the radiance reaching a particular pixel is

\[ L_{\text{pixel}} = \frac{d\Phi}{\sigma_{\text{pixel}} \cos \theta_{\text{pixel}}} \]  

(4.39)

where \( \Phi \) is the incoming light energy, \( \sigma_{\text{pixel}} \) is the solid angle subtended by the image plane pixel, and \( \theta_{\text{pixel}} \) is the angle between the connecting edge and the pixel normal \( n_{\text{pixel}} \). Figure 4.6 illustrates these values. This equation properly computes the light energy reaching the pixel regardless of the scaling of the image plane\textsuperscript{24}, so the \(|D|\) term is excluded during image combination. More precisely, by explicitly sampling a position on the pixel, the probability density of selecting a point is proportional to the total area of the image plane, canceling the \(|D|\) term in the integration. For more detail on this subject, see [DLW93].
4.2.6 Results

As figure 4.7 clearly illustrates, given the same amount of computational resources, bidirectional path tracing vastly outperforms path tracing in rendering complex lighting phenomena, particularly caustics. Since this is still a Monte Carlo algorithm, the error decreases proportionally to the root of the number of samples taken, which explains why the back surface in figure 4.7 appears noisier in the BDPT render. BDPT trades a small amount of efficiency in simple cases for greatly improved efficiency in complex ones. Modern renders often use complex specular surfaces, so this trade-off is generally worthwhile\textsuperscript{25}.

\textsuperscript{22}See [VG95].
\textsuperscript{23}The image plane size is arbitrary because, in a pinhole camera, each pixel maps to a specific set of directions. The image plane is never really considered, but a simulated image plane behind the pinhole can be scaled then moved closer or further from the pinhole while maintaining an identical field of view and pixel-to-solid angle mapping.
\textsuperscript{24}Notice that scaling the image plane and changing its distance to the pinhole proportionately does not affect any terms in the equation.
\textsuperscript{25}For more detailed discussion of BDPT performance, see [Vea98] and [VG95].
Chapter 5

Extensions to Bidirectional Path Tracing

In its common implementation, the bidirectional path tracing algorithm computes and caches an eye path and a light path, then connects the two. However, its mathematical description does not specifically imply that there is only one of each, or even that the precomputation needs to use paths specifically. Drawing from the ideas of distributed ray tracing, this chapter describes two modifications to bidirectional path tracing that take advantage of this ambiguity by gathering and combining more samples than simple paths provide.

5.1 Multipath Bidirectional Path Tracing

Since there is no requirement that only one eye and one light path be used, during a single pass, multipath bidirectional path tracing, or MBDPT, computes multiples of each and connects all possible pairs of eye and light paths. The evaluation of any given connection varies only in the path probability and weight computations, while the actual radiance contribution remains unchanged. More specifically, this algorithm is a modification to the standard bidirectional path tracing estimator and some additional accounting for the extra paths.
Figure 5.1: Multipath bidirectional path tracing with $n_l = 2$, $n_e = 1$ produces two explicit edges for the $s = 2$, $t = 1$ technique.

5.1.1 MBDPT Estimator

The new estimator for this algorithm takes an additional two parameters, the number of eye paths $n_e$ and the number of light paths $n_l$, resulting in the equation,

$$ F = \sum_{q=1}^{n_e} \sum_{r=1}^{n_l} \sum_{s \geq 0} \sum_{t \geq 0} w_{q,r,s,t}(\bar{x}_{q,r,s,t}) \frac{f_j(\bar{x}_{q,r,s,t})}{p_{q,r,s,t}(\bar{x}_{q,r,s,t})}. \quad (5.1) $$

While this estimator appears complex, in practice it amounts only to a selection of an eye-light path pair and a different computation of the weight and path probability terms. Otherwise, the majority of the calculation is identical to standard bidirectional path tracing. The $\alpha$ terms are precomputed and stored as before, then a multiplication is used to weight and connect eye and light prefixes.

The eye and light paths are all precomputed once, so there are $n_l + n_e$ total traces into the scene, but since all eye-light path pairs produce contributions, there are $n_l n_e$ total contributions. This algorithm significantly increases the correlation between samples, increasing variance, but the goal here is to have the number of contributions outpace this noise increase.

5.1.2 MBDPT Path Probability and Weighting

The real path probabilities are computed exactly as before, with the same $\alpha$ values as in normal bidirectional path tracing, since these are determined entirely by the random values generated during tracing. However, when computing potential probability densities during weighting, any given path vertex can be generated as one of multiple
paths. This means that the sampling techniques being considered are no longer simply a selection of eye and light vertex counts.

Having multiple paths means that the number of sampling techniques used for a single pixel increases proportionately to the product of the number of eye and light paths. For example, consider the situation illustrated in figure 5.1. Here there is a single eye path (consisting of vertex $x_4$) and two light paths, labeled $a$ and $b$. If only light path $a$ is considered, then the dotted line indicates the $s = 2, t = 1$ technique; however, since there are two light paths, this single technique becomes two: $s = 2a, t = 1$ and $s = 2b, t = 1$. If there were two eye paths, then the $s = 2, t = 1$ technique would expand into four techniques, allowing for all selections of eye and light paths.

These extra techniques are multiple importance sampled as normal in the weighting heuristic. As an example, the balance heuristic would be implemented as

$$w_{q,r,j}(\tilde{x}_{q,r,i}) = \frac{c_{q,r,i}p_{q,r,j}(\tilde{x}_{q,r,i})}{\sum_{q'=1}^{n_q} \sum_{r'=1}^{n_r} \sum_{j=1}^{s} c_{q',r',j}p_{q',r',j}(\tilde{x}_{q,r,i})},$$

(5.2)

where $\tilde{x}_{q,r,i}$ is the path using eye path $q$, light path $r$, $i$ light vertices, and $s + t - i$ eye vertices (to use notation similar to that seen in section 4.2.4). Again, while this appears to be complex, it is simplified significantly by the fact that which eye or light path a vertex lies on does not affect the probability of generating that vertex. This means that

$$p_{q,r,j}(\bar{x}) = p_{q',r',j}(\bar{x}) \quad \forall \bar{x},$$

(5.3)

where $j$ is a fixed path-length selection and $\bar{x}$ is a particular path. Using this fact, all that needs to be done in practice when computing a weight is to add potential path probabilities into the heuristic’s denominator $n_l \cdot n_e$ times.

### 5.1.3 Results

Multipath bidirectional path tracing was tested using a machine with an Intel Core i7 processor with 4 physical cores (8 logical) and a 2.86GHz clock rate. The test system has 16GB of DDR3 memory, far exceeding any requirements for the test scenes.

The algorithm was tested using the following scenes:

---

\(^1\)Letters are used here for clarity, though in the estimator equation 5.1 paths are referenced by index numbers $q$ and $r$.

\(^2\)Using a slight abuse of notation, $s = 2a$ indicates two vertices on light path $a$, $s = 2b$ two vertices on light path $b$, and so on.
(a) Indirect Scene Reference rendered with 1000 samples per pixel.
(b) Hard Scene Reference rendered with 5000 samples per pixel.

**Figure 5.2:** Reference renders were performed using standard bidirectional path tracing techniques with a very high number of samples per pixel, and are considered sufficiently-converged for testing purposes. Both images required over an hour of render time on the test machine.

- **Indirect Scene.** The first scene is a simple indirect-lighting test consisting of two regions separated by a wall with a gap. There is an area light in one region that is obscured in most of the second by the wall, meaning that lighting of the second region is due almost entirely to indirect lighting through the gap in the wall. All surfaces in the scene are diffuse. See figure 5.2a for a reference image.

- **Hard Scene.** The second scene consists of an area light placed extremely close to a diffuse surface, allowing light to only escape via the narrow gap between them. Below this arrangement is a perfect specular refractor with refractive index 1.45. This scene is designed to be a difficult test case for bidirectional path tracing and basically impossible for normal path tracing algorithms. This arrangement severely limits the number of useful paths, and is difficult to randomly sample into. See figure 5.2b for a reference image.

All renders in this section were made using the same codebase, simulating BDPT with
MBDPT set to only use one light path and one eye path.

Tables 5.1 and 5.2 show the performance of BDPT and MBDPT as functions of the number of samples per pixel, number of light paths, and number of eye paths. The performance is measured in terms of the computation time for each render and the mean squared error of the result when compared to the reference images seen in figure 5.2. Figure 5.3 plots these two values against each other, showing which algorithm performs more efficiently per-sample.

**Table 5.1**: Indirect Scene equal-time rendering results

<table>
<thead>
<tr>
<th>Samples Per Pixel</th>
<th>Eye Paths</th>
<th>Light Paths</th>
<th>Actual Time (s)</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>1</td>
<td>1</td>
<td>275.3</td>
<td>1146.03</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>368.5</td>
<td>1076.92</td>
</tr>
<tr>
<td>125</td>
<td>1</td>
<td>1</td>
<td>460.4</td>
<td>1031.32</td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>1</td>
<td>554.0</td>
<td>1003.44</td>
</tr>
<tr>
<td>175</td>
<td>1</td>
<td>1</td>
<td>641.8</td>
<td>981.74</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>1</td>
<td>732.4</td>
<td>966.51</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>5</td>
<td>156.2</td>
<td>1497.81</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>5</td>
<td>191.0</td>
<td>1403.12</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>5</td>
<td>233.5</td>
<td>1286.95</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>5</td>
<td>279.9</td>
<td>1222.49</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>5</td>
<td>311.1</td>
<td>1184.33</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>4</td>
<td>238.2</td>
<td>1753.1</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>4</td>
<td>279.1</td>
<td>1609.84</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>4</td>
<td>347.0</td>
<td>1465.18</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>4</td>
<td>418.4</td>
<td>1359.24</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>4</td>
<td>465.1</td>
<td>1309.01</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>3</td>
<td>257.7</td>
<td>1507.84</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>3</td>
<td>308.4</td>
<td>1402.7</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 5.1 – continued from previous page

<table>
<thead>
<tr>
<th>Samples Per Pixel</th>
<th>Eye Paths</th>
<th>Light Paths</th>
<th>Actual Time (s)</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>3</td>
<td>3</td>
<td>384.9</td>
<td>1297.14</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>3</td>
<td>462.2</td>
<td>1223.04</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>3</td>
<td>512.7</td>
<td>1191.09</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>2</td>
<td>230.6</td>
<td>1426.87</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>2</td>
<td>282.4</td>
<td>1336.45</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>2</td>
<td>345.2</td>
<td>1242.6</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>2</td>
<td>416.1</td>
<td>1174.35</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>2</td>
<td>459.7</td>
<td>1143.87</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1</td>
<td>156.2</td>
<td>1497.81</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>1</td>
<td>191.0</td>
<td>1403.12</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>1</td>
<td>233.5</td>
<td>1286.96</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>1</td>
<td>279.9</td>
<td>1222.49</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>1</td>
<td>311.1</td>
<td>1184.34</td>
</tr>
</tbody>
</table>

### Table 5.2: Hard Scene equal-time rendering results

<table>
<thead>
<tr>
<th>Samples Per Pixel</th>
<th>Eye Paths</th>
<th>Light Paths</th>
<th>Actual Time (s)</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1</td>
<td>1</td>
<td>30.1</td>
<td>3573.96</td>
</tr>
<tr>
<td>75</td>
<td>1</td>
<td>1</td>
<td>44.9</td>
<td>3454.7</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>59.4</td>
<td>3332.84</td>
</tr>
<tr>
<td>125</td>
<td>1</td>
<td>1</td>
<td>75.1</td>
<td>3260.67</td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>1</td>
<td>90.3</td>
<td>3214.41</td>
</tr>
<tr>
<td>175</td>
<td>1</td>
<td>1</td>
<td>102.9</td>
<td>3172.18</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>1</td>
<td>117.3</td>
<td>3128.15</td>
</tr>
<tr>
<td>225</td>
<td>1</td>
<td>1</td>
<td>131.3</td>
<td>3100.7</td>
</tr>
</tbody>
</table>

Continued on next page
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Samples Per Pixel</th>
<th>Eye Paths</th>
<th>Light Paths</th>
<th>Actual Time (s)</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>1</td>
<td>1</td>
<td>146.0</td>
<td>3081.72</td>
</tr>
<tr>
<td>275</td>
<td>1</td>
<td>1</td>
<td>160.5</td>
<td>3072.13</td>
</tr>
<tr>
<td>300</td>
<td>1</td>
<td>1</td>
<td>177.7</td>
<td>3056.23</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>5</td>
<td>22.7</td>
<td>3607.31</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>5</td>
<td>27.2</td>
<td>3539.66</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>5</td>
<td>33.7</td>
<td>3490.94</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>5</td>
<td>40.7</td>
<td>3412.09</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>5</td>
<td>44.9</td>
<td>3384.8</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>5</td>
<td>68.0</td>
<td>3240.15</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>5</td>
<td>89.2</td>
<td>3156.01</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>5</td>
<td>111.8</td>
<td>3110.53</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>4</td>
<td>30.3</td>
<td>3718.19</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>4</td>
<td>36.3</td>
<td>3654.08</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>4</td>
<td>45.1</td>
<td>3549.7</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>4</td>
<td>54.3</td>
<td>3462.37</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>4</td>
<td>60.2</td>
<td>3418.68</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>4</td>
<td>91.6</td>
<td>3252.39</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>4</td>
<td>121.3</td>
<td>3165.28</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>4</td>
<td>151.1</td>
<td>3123.62</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>3</td>
<td>33.2</td>
<td>3882.62</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>3</td>
<td>40.1</td>
<td>3767.68</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>3</td>
<td>49.7</td>
<td>3664.27</td>
</tr>
<tr>
<td>18</td>
<td>3</td>
<td>3</td>
<td>59.8</td>
<td>3578.56</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>3</td>
<td>66.1</td>
<td>3524.95</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>3</td>
<td>99.3</td>
<td>3330.89</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
<td>3</td>
<td>134.2</td>
<td>3224.42</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>3</td>
<td>167.1</td>
<td>3160.73</td>
</tr>
</tbody>
</table>

Continued on next page
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Samples Per Pixel</th>
<th>Eye Paths</th>
<th>Light Paths</th>
<th>Actual Time (s)</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>2</td>
<td>31.7</td>
<td>4202.04</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>2</td>
<td>38.6</td>
<td>4047.25</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>2</td>
<td>47.2</td>
<td>3894.85</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>2</td>
<td>57.5</td>
<td>3784.37</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>2</td>
<td>63.4</td>
<td>3719.4</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>2</td>
<td>95.3</td>
<td>3492.29</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>2</td>
<td>127.3</td>
<td>3342.37</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>2</td>
<td>157.4</td>
<td>3272.04</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1</td>
<td>25.6</td>
<td>5060.45</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>1</td>
<td>31.1</td>
<td>4797.04</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>1</td>
<td>37.9</td>
<td>4493.52</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>1</td>
<td>45.5</td>
<td>4281.92</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>1</td>
<td>50.5</td>
<td>4182.65</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>1</td>
<td>75.9</td>
<td>3895.68</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>1</td>
<td>101.9</td>
<td>3700.89</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>1</td>
<td>126.0</td>
<td>3567.51</td>
</tr>
</tbody>
</table>
Figure 5.3: Efficiency curves showing reduction of error with render time of BDPT versus different settings of MBDPT.
The results shown here indicate that the performance of MBDPT depends heavily on the scene being rendered. It was easily outperformed by BDPT in the indirect scene, where all light interactions were simple Lambertian diffuse bounces, with a trend showing eye paths to be relatively more important to the quality of the image. The results for the hard test scene show a small improvement when MBDPT uses significantly more light paths than eye paths, likely due to the presence of specular surfaces and caustics. All versions of this algorithm show the same $\sqrt{N}$ reduction in error, indicating that they asymptotically converge to roughly the same performance, at least in the cases tested here.

**Computational Complexity**

Unlike originally assumed, this algorithm generally does not scale linearly with the number of paths used during each trace. Instead, shadow tests performed during subpath connection quickly dominate computation time, making the performance of the algorithm roughly

$$O(n_l L_l n_e L_e),$$ \hspace{1cm} (5.4)

where $n_l$ and $n_e$ are the number of light and eye paths, respectively, and $L_l$ and $L_e$ are the lengths of these paths. This is a product of having to test for shadowing between every pair of eye and light vertices, of which there are $n_e L_e$ and $n_l L_l$, respectively.

### 5.2 Branching Bidirectional Path Tracing Proposal

A more direct application of the ideas of distributed path tracing to bidirectional path tracing is the use of branching paths or trees instead of normal paths, as illustrated in figure 5.4. The idea here is to branch at surfaces where the material is described by a complex BRDF to perform multisampling at those points specifically, rather than relying on later rays to hit similar points. I propose that this change could increase the efficiency of computations of long light paths in scenes with complex materials, though the benefit would likely depend on the scene and materials in question. Additionally, the degree of branching at any given bounce of a path can be controlled probabilistically, allowing for
Figure 5.4: Branching bidirectional path tracing uses trees instead of paths, so different techniques can be used to create varying numbers of samples. This is an illustration of multiple \( s = 3, t = 4 \) technique samples.

importance sampling. This algorithm, however, is much more complex than MBDPT and is left for future work.
Chapter 6

Conclusion

This thesis investigated the viability of a modification to bidirectional path tracing, called multipath bidirectional path tracing, where multiple path prefixes and suffixes are used to gather more data from a limited amount of precomputation. As was demonstrated, this modification has very limited utility, and only outperforms bidirectional path tracing in certain cases. The performance improvement occurs only when the algorithm gathers more samples from complex light interaction than BDPT. This leads to the conclusion that MBDPT in form presented here is not a viable alternative to traditional BDPT due to increased code complexity with little benefit.

6.1 Future Work

It is still possible that MBDPT can show good performance if the cost of shading intersection tests can be minimized, since these dominate the algorithm’s computation time. Also, as was discussed earlier, a branching variant of bidirectional path tracing is a more direct and controllable method of gathering additional samples, and remains a possible avenue for research.
Appendix A

Probability Density Computation

A probability density function (or PDF) is defined as the relative probability of a random variable taking on a particular value, and when it is integrated of a particular range, provides the probability that the random variable takes on a value in that range. Monte Carlo methods depend on being able to compute PDF values whenever samples are taken, and more complex algorithms optimize performance by tuning the density functions themselves (as in importance sampling). This appendix describes practical approaches to computing and selecting probability density functions for several common situations encountered in physically based rendering.

A.1 Probability Density Measurement and Sampling

Probability density is defined as a function that, when integrated, provides the probability of a random variable $X$ taking on some value $x$ within the domain of integration,

$$Pr[a \leq X \leq b] = \int_a^b p_X(x) \, d\mu(x),$$

(A.1)

where $p_X$ is the PDF of random variable $X$ and $\mu$ is a measure\(^1\) on the domain of $X$. In rendering, $\mu$ is generally either the solid angle $\sigma$, projected solid angle $\sigma^\perp$, or area $A$ measure.

\(^1\)Roughly speaking, a measure is the n-dimensional volume of a subset of the domain of $x$. 
The formal definition of the PDF is not particularly useful for Monte Carlo rendering since \( p_X \) is generally the unknown being calculated. However, finding the cumulative distribution function (or CDF) of \( X \),

\[
Pr[X < x] = P_X(x) = \int_{-\infty}^{x} p_X(x') \, d\mu(x'), \tag{A.2}
\]

allows us to explicitly define \( p_X \),

\[
p_X(x) = \frac{dP_X}{d\mu}(x). \tag{A.3}
\]

In rendering, the PDF will often take on one of the forms,

\[
p(x) = \frac{dP}{d\sigma}(x) \quad \text{or} \quad \frac{dP}{d\sigma^\perp}(x) \quad \text{or} \quad \frac{dP}{dA}(x), \tag{A.4}
\]

or a combination thereof.

### A.1.1 Sampling Arbitrary PDFs

Computing probability densities is necessary for Monte Carlo integration, but to have effective importance sampling, it is necessary to be able to create sampling techniques based on existing probability densities\(^2\).

#### 1D Inversion Method

Sampling a one-dimensional probability distribution can be done using the inversion method\(^3\), which defines a function mapping a uniformly-distributed random variable \( U \in [0, 1] \) to the possible values of \( X \). Without getting into the mathematical details yet, this is done by finding the CDF, inverting it, and gathering samples as a function of \( U \).

For example, suppose we want to gather samples of \( X \in [0, 2] \) according to a probability density \( p_X(x) = \frac{x}{2} \) using the uniformly-distributed random variable \( U \in [0, 1] \). Finding the CDF of \( X \) is done by integrating, as in equation A.2,

\[
P_X(x) = \int_{0}^{x} \frac{x'}{2} \, dx' = \left( \frac{x^2}{4} \right)_{x'=0}^{x} = \frac{x^2}{4}.
\]

\(^2\)Much of the material in this section is drawn from [PH10], which also contains a number of examples not covered here. However, as a warning, [PH10] is very technical in nature, and basic principles may be difficult to learn from it.

\(^3\)See [PH10, p.643]
Set $u = \text{CDF}(x)$

**Figure A.1:** Example cumulative density function for the PDF $p_X(x) = 1/2$, and an illustration of how 1D inversion works.

Inverting this equation gives us the function on samples of $U$,

$$ x = P_X^{-1}(u) = 2 \sqrt{u}, $$

where the values of $u$ are sampled according to the PDF $p_U(u) = 1$. Since uniformly-distributed random variables are trivial to sample, this allows us to sample $X$ from $p_X(x)$ directly. We’ll prove this result shortly.

**Generalized Inversion Method**

Suppose we have a random variable $X$ that has been sampled from some PDF $p_X(x)$ and a one-to-one function $Y = y(X)$ on that random variable. Given the distribution $p_X(x)$, it is possible to find the distribution $p_Y(y)$, and by selecting $X$ and $Y$ carefully, this technique can be used to sample any well-defined, differentiable PDF.

Since $y(x)$ is one-to-one, it is a strictly increasing or decreasing function, meaning its derivative is strictly greater than or less than zero. Therefore, assuming a strictly increasing function, if $x_1 \leq x_2$, then $y_1 = f(x_1) \leq f(x_2) = y_2$, implying

$$ Pr[Y \leq y(x)] = Pr[X \leq x], $$

which in turn means that the CDFs of the two variables are equal, when measured with respect to their own domains,

$$ P_Y(y) = P_Y(y(x)) = P_X(x). \quad (A.5) $$
Using the definition of the CDF (equation A.2),

\[ P_Y(y) = \int_{-\infty}^{y} p_Y(y')dy' = \int_{-\infty}^{x} p_X(x')dx' = P_X(x). \] (A.6)

Differentiating then yields

\[ p_Y(y)dy = p_X(x)dx, \] (A.7)

which can be rearranged into an equation for converting PDFs\(^4\),

\[ p_Y(y) = \left| \frac{dy}{dx} \right|^{-1} p_X(x), \] (A.8)

with the absolute value being a result of \( y(x) \) possibly having a negative derivative (in the case of a strictly decreasing function).

Going back to the earlier example, \( p_U(u) = 1 \) and the function we found for converting the PDFs is \( x = 2 \sqrt{u} \). Using equation A.8, we find

\[
\begin{align*}
p_X(x) &= \left| \frac{dx}{du} \right|^{-1} p_U(u) \\
&= \left| 2 \frac{d}{du} \sqrt{u} \right|^{-1} \\
&= \left| 2 \frac{1}{2 \sqrt{u}} \right|^{-1} \\
&= \sqrt{u} = \sqrt{\left(\frac{x^2}{2}\right)} = \frac{x}{2},
\end{align*}
\]

thus verifying the result found earlier.

All that equation A.8 requires is that the CDFs of the two variables be equal as in equation A.5, which yields the transformation\(^5\)

\[ y(x) = P_Y^{-1}(P_X(x)). \] (A.9)

This is a generalization of the inversion method seen in section A.1.1, and it is easy to verify that the two methods agree.

As described in [PH10, p.661], this inversion technique also works with multi-dimensional random variables, and a similar proof yields

\[ p_Y(y) = p_Y(T(x)) = \frac{p_X(x)}{|J_T(x)|}, \] (A.10)

\(^4\)See [PH10, p.660].
\(^5\)See [PH10, p.661].
where \( Y = T(X) \) is a bijection between the two random variables \( X \) and \( Y \), and \( |J_T(x)| \) is the absolute value of the determinant of the Jacobian matrix,

\[
\begin{pmatrix}
\frac{\partial T_1}{\partial x_1} & \cdots & \frac{\partial T_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial T_n}{\partial x_1} & \cdots & \frac{\partial T_n}{\partial x_n}
\end{pmatrix},
\]

(A.11)

where \( T(x) = (T_1(x), \ldots, T_n(x)) \).

### A.1.2 Constant-Density Sampling Examples

All of the above equations are useful when a random variable with a particular distribution is provided, but it is not immediately obvious how they are helpful when determining how to sample a particular domain. To this end, the following examples demonstrate how to sample some common domains of integration with constant PDFs corresponding to the three measures \( \sigma, \sigma^\perp \), and \( A \).

#### Uniform Hemisphere Sampling

There are many cases in which uniformly sampling a sphere is useful or necessary in rendering (texturing or sampling diffuse materials, for example), but doing so correctly is not as straightforward as it seems.

Sampling the surface of a hemisphere is most easily done with spherical coordinates \( \theta, \phi \), and \( r \), setting \( r = 1 \) generally. The simplest approach is to sample \( \theta \) and \( \phi \) uniformly over their domains:

\[
\begin{align*}
\theta &= \frac{\pi}{2} u \\
\phi &= 2\pi v,
\end{align*}
\]

(A.12)

where \( u \), and \( v \) are samples from uniformly-distributed random variables \( U, V \in [0, 1] \).

To determine if this is the correct approach, we need to find the probability density with respect to solid angle \( p_\sigma(\omega) \), which doesn’t have any of the orientation concerns of spherical coordinates. To do so, we first consider how to measure the probability of a direction being chosen in a particular region of the hemisphere \( \Omega \),

\[
Pr[\omega \in \Omega] = \int_\Omega p_\sigma(\omega')d\sigma(\omega') = \int_\Omega p_{\theta,\phi}(\theta', \phi')d\theta(\theta')d\phi(\phi'),
\]

(A.13)
where $p_{\theta,\phi}$ is the joint probability distribution of choosing a particular pair of $\theta$ and $\phi$ values. Since $\Omega$ is arbitrary, and can be arbitrarily small, this implies that

$$p_{\sigma}(\omega')d\sigma(\omega') = p_{\theta,\phi}(\theta', \phi')d\theta(\theta')d\phi(\phi').$$  \hspace{1cm} (A.14)$$

Using the relation between differential solid angle and differential spherical coordinates, $d\sigma(\omega) = \sin \theta' d\theta(\theta')d\theta(\theta')$, (leaving out some symbols for brevity) this equation yields

$$p_{\sigma} = \frac{1}{\sin \theta} p_{\theta,\phi} = \frac{1}{\sin \theta} p_{\theta} p_{\phi}.$$  \hspace{1cm} (A.15)$$

where $p_{\theta,\phi}$ is separable since these angles are chosen separately to begin with. Now, using equation A.8, we can find the PDF with respect to solid angle,

$$p_{\sigma} = \frac{1}{\sin \theta} p_{\theta} p_{\phi} = \frac{1}{\sin \theta} \left| \frac{d\theta}{du} \right|^{-1} p_u \left| \frac{d\phi}{dv} \right|^{-1} p_v$$

$$= \frac{1}{\sin \theta} \left( \frac{2}{\pi} \right) \left( \frac{1}{2\pi} \right) (1) (1)$$

$$= \frac{1}{\pi^2 \sin \theta}.$$  \hspace{1cm} (A.16)$$

This PDF is obviously not uniform, and in fact skews samples towards the apex of the hemisphere (where there is a singularity in spherical coordinates). To find the correct sampling method, we have to work from the other direction and start from the desired constant PDF $p_{\sigma}(\omega) = c$.

By definition, a PDF has to integrate to 1 over its domain, which allows us to find a constant pdf over the hemisphere:

$$1 = \int_{H^+} p_{\sigma}(\omega) d\sigma(\omega) = c \int_{H^+} d\sigma(\omega) = c(2\pi) \Rightarrow c = p_{\sigma}(\omega) = \frac{1}{2\pi}.$$  \hspace{1cm} (A.17)$$

Using equation A.15, this can be converted to spherical coordinates:

$$p_{\theta,\phi} = \frac{\sin \theta}{2\pi}.$$  \hspace{1cm} (A.17)$$

Since it was derived from a PDF with a different measure, $p_{\theta,\phi}$ is not trivially separable. Instead, to sample this distribution, the marginal density $p_{\theta}(\theta)$ is sampled first, then the conditional density $p_{\phi}(\phi|\theta)$ is sampled second, potentially based on the value of the first sample.
First, finding the marginal density is done by "integrating out" the other variable,

\[ p_\theta(\theta) = \int_0^{2\pi} p_{\theta,\phi}(\theta, \phi') d\phi' = \int_0^{2\pi} \frac{\sin \theta}{2\pi} d\phi' = \sin \theta. \] (A.18)

Then, the conditional density is found by using the definition,

\[ p(a|b) = \frac{p(a,b)}{p(b)}, \] (A.19)

which leads to,

\[ p_\phi(\phi|\theta) = \frac{p_{\theta,\phi}(\theta, \phi)}{p_\theta(\theta)} = \frac{\frac{\sin \theta}{2\pi}}{\sin \theta} = \frac{1}{2\pi}. \] (A.20)

Now, all that’s left to do is perform a 1D inversion on each of these densities.

\[ P_\theta(\theta) = \int_0^\theta \sin \theta' d\theta(\theta') = (\cos \theta')_{\theta'=0}^\theta \]
\[ = 1 - \cos \theta \]
\[ P_\phi(\phi|\theta) = \int_0^\phi \frac{1}{2\pi} d\phi(\phi') \]
\[ = \frac{1}{2\pi} \phi, \] (A.21)

therefore,

\[ \theta = P_\theta^{-1}(P_u(u)) = P_\theta^{-1}(u) = \arccos(1 - u) \iff \theta = \arccos u \]
\[ \phi = P_\phi^{-1}(P_v(v)) = P_\phi^{-1}(v) = 2\pi v, \] (A.22)

which can be easily be demonstrated to generate a uniform distribution over the hemisphere.

**Cosine-Weighted Hemisphere Sampling**

When importance sampling the rendering equation, or BRDFs in general, it is often useful to have a cosine-weighted distribution on the hemisphere, meaning that samples are distributed proportionally to the cosine of \( \theta \). Exactly like in the case of
uniform sampling, we begin by finding the PDF with respect to solid angle:

\[ p_\sigma(\omega) \propto \cos \theta \]
\[ p_\sigma(\omega) = c \cos \theta \]

\[ 1 = \int_{H^+} c \cos \theta d\sigma(\omega) \]
\[ = c \int_0^{\pi/2} \int_0^{2\pi} \cos \theta' \sin \theta' d\phi'(\theta') d\theta'(\theta') \]
\[ = 2\pi c \left( \frac{\sin^2 \theta}{2} \right) \bigg|_{\theta=0}^{\theta=\pi/2} = \pi c \]
\[ \Rightarrow p_\sigma(\omega) = \frac{\cos \theta}{\pi}. \]

Once this is found, the procedure for finding a sampling technique is identical to the example for uniform hemisphere sampling, so it will not be reproduced here.

Of note, however, is that this is still a constant PDF when measured with respect to projected solid angle.

\[ p_\sigma(x) = \frac{dP}{d\sigma}(x) \]
\[ p_\sigma^\perp(x) = \frac{dP}{d\sigma^\perp}(x) = \frac{d\sigma}{d\sigma^\perp} \frac{dP}{d\sigma}(x) = \frac{1}{\cos \theta} p_\sigma(x), \quad (A.23) \]

which means that the cosine-weighted hemisphere sampling, when measured with respect to projected solid angle, is \( p_\sigma^\perp(\omega) = \frac{1}{\pi} \). This makes cosine-weighted hemisphere sampling extremely useful when using the rendering equation defined with respect to projected solid angle.

**Area Sampling**

Area sampling is the selection of a point on a particular bounded surface, and the simplest version of this is uniform area sampling where a point is chosen with equal likelihood over the entire surface (specifically, with PDF \( p_A(x) = \frac{1}{A_{\text{surface}}} \)). Techniques for uniformly sampling different shapes are very well documented, so they will not be reproduced here.

There exist cases when area sampling is useful, but incompatible with the domain of integration of a Monte Carlo algorithm. A good example of this is using area sampling for direct lighting during path tracing, which is an important optimization for
quickly rendering area lights. Path tracing is based on the hemispherical formulation of the rendering equation, so its domain is measured with respect to solid angle, and any PDF value computed needs to also be measured with respect to solid angle\(^6\). This is the opposite situation to that in section 4.2.2, but the solution is the same. To this end, it is possible to convert an area sampling PDF to a solid angle measure PDF, as would be sampled from some particular point.

Suppose a point \(x\) is sampled uniformly on a surface \(S\), and that the normal at that point is \(N_x\). This point could potentially be sampled from another point \(y\) via hemispherical sampling, but the domain for sampling exclusively from \(S\) can be impossibly complex, so directly finding the solid angle PDF \(p_\sigma (y \to x)\) is likewise impossible in the general case. However, by using equation 2.3, we can easily derive a conversion between these measures:

\[
P_A = \frac{dP}{dA} = \frac{dP}{d\sigma} \frac{d\sigma}{dA} = \frac{dP}{d\sigma} \frac{\cos \theta}{r^2} = p_\sigma \frac{|N_x \cdot \omega|}{\|x - y\|^2},
\]

where \(\omega\) is the direction from \(x\) to \(y\). It is similarly possible to derive the conversion between area and projected solid measures this way, which conveniently yields the geometry term (equation 2.11),

\[
P_A = \frac{dP}{dA} = \frac{dP}{d\sigma^\perp} \frac{d\sigma}{d\sigma^\perp} = \frac{dP}{d\sigma^\perp} \frac{\cos \theta_i}{r^2} = p_{\sigma^\perp} G(x, y).
\]

### A.2 Diffuse Surfaces and Emitters

Diffuse surfaces are generally the easier type to handle in any physically-based renderer due to having well-defined BRDFs. In the case of simpler BRDFs, for example the Lambertian \(f_s(x, \omega_i, \omega_o) = \rho / \pi\), it is possible to fix the position and incoming direction variables, then use the BRDF as a PDF for importance sampling. In the Lambertian case, \(^6\)Note that generally path tracing skips this complication by using the area formulation for the direct lighting contribution, and the hemispherical formulation for everything else. This is a perfectly valid approach, since each bounce is strictly speaking a separate Monte Carlo evaluation, but when comparing probabilities for multiple importance sampling of direct lighting, all probabilities have to be measured in the same space, so such a simplification is impossible.
this amounts to hemispherical sampling (or cosine-weighted sampling to importance sample the rendering equation as a whole). More complex BRDFs are harder to directly importance sample, though it is sometimes possible to use histogram-based methods to do so approximately.

A.2.1 Diffuse Surface Probability

This thesis discusses two methods of sampling to and from diffuse surfaces, area and hemisphere sampling, and this section will discuss practical approaches to using these methods.

Area Sampling the Scene

To area sample a point in a scene, it is necessary to make two selections: which surface to sample and what point to sample on it. The first can be done a number of ways, for example by randomly choosing any polygon in the scene. This is not an optimal approach since the sampling gives equal weight to very small and very large polygons, which can cause a considerable amount of variance. To importance sample surfaces, it is instead necessary to weight their selection probabilities by their surface areas, leading to a constant sampling PDF $p_A(x) = \frac{1}{S_{A_{total}}}$ across the entire set of surfaces being sampled.

Doing so requires us to change the set of surface areas in the scene into a discrete PDF, and using inversion to sample from it. Finding the discrete surface selection PDF is easy and it can be defined as

$$p_{sel}(i) = \frac{S_{A_i}}{S_{A_{total}}},$$  \hspace{1cm} (A.26)

where $i$ is the index of a particular surface, and $S_{A_i}$ is the area of that surface. The inversion step is slightly more difficult. Suppose that the surface selection PDF is stored as an array of values and we wish to sample according to it. First, the CDF is computed by finding the exclusive prefix sum for every element in the array, meaning the sum of that element’s preceding elements. Once this is done, we know that the prefix sum array is already sorted, so sampling is done by selecting a random number $u \in [0, 1]$, then binary searching the prefix sum array for the largest value less than $u$. The index $i$ of
this value is then the selected surface, and $p_{sel}(i)$ is the sampling PDF that should be used in any Monte Carlo evaluation.

Once this is done, selecting a point on the surface depends on what what shape it is, and many techniques for different surfaces are easy to find references for. Supposing that this is done uniformly, the PDF of the selected point $x$ is $p_{A,i}(x) = 1/S_{A,i}$ where $S_{A,i}$ is the area of the surface $x$ lies on. This yields the original sampling PDF we were looking for,

$$p_A(x) = p_{sel}(i)p_{A,i} = \frac{1}{S_{A_{\text{total}}}}. \tag{A.27}$$

**Hemisphere Sampling From a Diffuse Surface**

Since a diffuse surface has a BRDF with at least one well-defined component, sampling a direction is done using one of the hemisphere sampling methods mentioned above, or by normalizing and inverting some approximation of the BRDF.

**Path Probability Density on Diffuse Interactions**

Suppose that a path $\bar{x} = x_0 \ldots x_i$ with a PDF $p_\mu(\bar{x})$ (measured with respect to the area product measure $\mu$) has been sampled, and we wish to extend this path by adding a new vertex $x_{i+1}$ to create $\bar{x}' = x_0 \ldots x_i x_{i+1}$.

Area sampling the new vertex is mathematically easier, but may fail if there is a surface obscuring $x_{i+1}$ from $x_i$. However, if there is no such surface, the PDF of the new path is computed simply multiplying in the area PDF of sampling $x_{i+1}$,

$$p_\mu(\bar{x}') = p_\mu(\bar{x})p_A(x_{i+1}). \tag{A.28}$$

Hemisphere sampling is generally used instead since it is guaranteed to generate a valid new vertex\(^7\), though updating the path PDF is more complex. Suppose a direction $\omega$ is chosen at point $x_i$ with PDF $p_\sigma(\omega)$, and ray casting finds the point $x_{i+1} = r_M(x_i, \omega)$. The hemispherical PDF cannot be used directly to update the path PDF since it doesn’t match the form of the area product measure. Instead, equation A.24 is used to find the area-measure PDF, which, like before, can be simply multiplied in,

$$p_\mu(\bar{x}') = p_\mu(\bar{x})p_{\sigma\rightarrow A}(x_{i+1}) = p_\mu(\bar{x})p_\sigma(\omega) \frac{|N_{x_{i+1}} \cdot \omega|}{\|x - y\|^2}. \tag{A.29}$$

\(^7\)Or none at all if its sampled ray leaves the scene.
Light Path Initial Probability Density

When creating a light path, the path must be initialized with a first vertex $x_0$. When only finite-area lights are present, this process is identical to the scene area sampling discussed above. Point lights, however, can present a problem since they don’t have a well-defined surface area and will be discussed in the next section.

A.3 Specular Surfaces and Emitters

A specular surface is defined as having a physical description that includes the Dirac delta function $\delta$, which means that it is not well-defined and sampling it becomes more complex. Ideal reflectors and refractors are examples of specular surfaces because their BRDFs contain Dirac delta functions, which are necessary to mathematically define a single exact outgoing direction for any given incoming direction. The PDF of any sampling that involves a specular surface generally also contains delta functions. Delta functions can also describe the dimensions of an object, and often do so in the cases of point lights and pinhole camera models.

A.3.1 Dirac Delta

The Dirac delta is not a function by formal definition, but is useful to treat as such, as long as it is not actually evaluated. Specifically, the delta function is described in two parts:

$$
\delta(x) = \begin{cases} 
+\infty & \text{if } x = 0 \\
0 & \text{if } x \neq 0
\end{cases} \quad (A.30a)
$$

$$
1 = \int_{-\infty}^{\infty} \delta(x) \, dx. \quad (A.30b)
$$

By this description, the delta function is an infinitesimally-thin, infinitely-tall vertical spike at $x = 0$, with an area of 1 underneath it. The reason it exists is because it exhibits the useful property that

$$
\int_{-\infty}^{\infty} f(x)\delta(x - c) \, dx = f(c), \quad (A.31)
$$

allowing it to be used to "extract" values of a function during integration.
For this reason, it is used to describe, for example, the BRDF of an ideal reflector, where for any incoming direction there exists only one precise outgoing direction. The delta function is the only means by which such a BRDF still integrates to \([0, 1]\) in a consistent manner, making the BRDF

\[ f_s(x, \omega_i, \omega_o) = f(\omega_i) \delta(\text{Ref}_{x}(N_x, \omega_i) - \omega_o), \]

where \(f(\theta_i)\) is some scalar reflectivity function (i.e. the Fresnel equation), and \(\text{Ref}(N, \theta)\) represents the function that reflects direction \(\theta\) over the surface normal \(N\).

### A.3.2 Specular Surfaces

Physically-based specular surfaces are those that exhibit reflection, and if transparent, refraction. Common examples are mirrors, glass, and water. All these materials reflect and refract based on the Fresnel equations (assuming they are non-magnetic, in which case the equations become much more complex).

The Fresnel equations are a set of equations that give the fraction of light \(R\) that is reflected by a surface, and conservation of energy give the fraction transmitted through the surface, \(T = 1 - R\). There exists one equation for light polarized parallel to the surface,

\[ R_p = \frac{\left| n_1 \cos \theta_i - n_2 \theta_t \right|^2}{\left| n_1 \cos \theta_i + n_2 \theta_t \right|}, \quad (A.32) \]

where \(n_1\) is the refractive index of the material outside the surface, \(n_2\) is the refractive index of the material inside the surface, and \(\theta_i, \theta_t \in [0, \pi/2]\) are the incident and transmitted angles relative to the surface normal\(^8\). Another equation describes light polarized perpendicular to the surface,

\[ R_s = \frac{\left| n_1 \cos \theta_i - n_2 \theta_t \right|^2}{\left| n_1 \cos \theta_i + n_2 \theta_t \right|}. \quad (A.34) \]

If the light is unpolarized, the result from these equations is simply their average,

\[ R = \frac{R_p + R_s}{2}. \quad (A.35) \]

---

\(^8\) These angles can be found using Snell’s law

\[ \frac{\sin \theta_i}{\sin \theta_t} = \frac{n_2}{n_1}. \quad (A.33) \]
The reason that these equations are discussed here is because the fraction of light that is reflected versus refracted can be, and in the case of path tracing has to be, randomly sampled. This is one common source of non-delta components of specular BRDFs since it amounts to a single binary choice of whether to reflect or not. This yields a very easily-sampled discrete PDF,

\[ p(\text{reflect}) = R \]
\[ p(\text{transmit}) = T = 1 - R. \]

Given a purely-specular, refractive surface, for any given incoming direction \( \omega_i \), there are exactly one or two\(^9\) outgoing directions \( \theta_r \) and possibly \( \theta_t \), as determined by the reflection formula and Snell’s law, respectively. Combining equations A.30, A.32, and A.34, we can completely describe this surface’s BRDF for unpolarized light:

\[
f_s(x, \omega_i, \omega_o) = \begin{cases} 
\left( \frac{R_p + R_s}{2} \right) \delta(\omega_o - \omega_r) + \left( 1 - \frac{R_p + R_s}{2} \right) \delta(\omega_o - \omega_t) & \text{if } \omega_t \text{ exists,} \\
\delta(\omega_o - \omega_r) & \text{otherwise.}
\end{cases}
\]

(A.37)

Note that this BRDF only evaluates to 0 or \(+\infty\) (possibly with some scaling factors). The reason that this doesn’t break down in Monte Carlo integration is because the hemispherical-sampling PDF is defined similarly,

\[
p(\omega_o) = P_r \delta(\omega_o - \omega_r) + P_t \delta(\omega_o - \omega_t),
\]

(A.38)

where \( P_r \) and \( P_t \) are the probabilities of reflecting and transmitting, respectively. Monte Carlo methods divide the evaluated function by the probability density of the sample taken, meaning that the delta functions cancel out, leaving only, in this example, the Fresnel scaling terms. In path tracing, this cancellation always happens, but in bidirectional path tracing, for some parts of the evaluation, this cannot be assumed to be the case.

\(^9\)When leaving a material with a higher index of refraction, beyond the critical angle

\[
\theta_c = \arcsin \frac{n_2}{n_1},
\]

(A.36)

light will undergo total internal reflection, and there will be no transmitted ray.
A.3.3 Specular Emitters

Emitters can be specular in two senses. By far the most common case is the diffuse point light, where the selection of a point on the light’s surface can only be defined by a Dirac delta,

\[
L_e(x, \omega_o) = I \delta(x - x_p),
\]

\[
P_A(x) = \delta(x - x_p)
\]

(A.39)

where \(x_p\) is the position of the point light and \(I\) is a constant intensity value\(^{10}\). As before, these delta terms generally cancel each other out since it is impossible to "hit" a point light with a random ray, meaning all point lights have to be explicitly sampled.

The other form of specular emitter is a laser-like emitter, where the outgoing direction is defined by a delta function,

\[
P_\sigma(\omega_d) = \delta(\omega_o - \omega_d),
\]

(A.42)

where \(\omega_d\) is the direction in which the emitter is pointing. This is a far more uncommon light, but is generally handled in the same way that any directionally-specular surface would be, and will not be covered in detail here.

A.3.4 Handling Specular PDFs

Handling delta functions in a practical setting is extremely difficult to do directly, particularly because floating point errors can easily cause two mathematically-identical

\(^{10}\)As an aside, the reason for the intensity value is that there cannot actually be an emitted radiance value defined for a point light since it lacks any surface area and normal direction. Instead, it is possible to compute its radiant intensity in \(\text{W/sr}\),

\[
I = \frac{\Phi}{4\pi}
\]

(A.40)

which describes the total emitted energy \(\Phi\) distributed over the entire sphere. Using equations 2.1 and 2.13, the incoming power at an infinitesimal patch \(dA\) at point \(y\) can be found,

\[
d\Phi_{i,y} = Id\sigma = \frac{\Phi dA \cos \theta}{4\pi \frac{r^2}}
\]

where \(\theta\) is the angle between the light direction and the surface normal at \(y\), and \(r^2\) is the distance between \(y\) and the light [Gig14]. This leads directly to the irradiance at \(y\),

\[
E = \frac{d\Phi_{i,y}}{dA} = \frac{\Phi_{\text{light}} \cos \theta}{4\pi r^2}.
\]

(A.41)
delta functions to mismatch. In the context of rendering, it is better to simply keep track of what the delta functions represent, and set them to zero unless they are "known" to be satisfied. For this reason, sampling a specular PDF cannot be done with area sampling, because even though it is possible to artificially choose a point along a specular direction, such a point could not realistically be chosen by random sampling on surfaces. Similarly, it is impossible to explicitly store the PDF when using hemispherical sampling since it contains a delta function.

The solution to these problems is generally to store a flag that indicates whether a particular value contains a specular term, and to set the value to zero in any evaluation that does not "know" that the delta function should be satisfied. Other than the specular flag, a stored PDF value still has to record non-specular terms, such as the probability of reflecting versus refracting at a surface.

The best example of this handling is in potential path evaluation for weighting in bidirectional path tracing (see section 4.2.4). Suppose that the path being evaluated contains a single specular vertex \( x_s \) that is not the endpoint of either the light path or eye path (as illustrated in figure A.2). This means that there exist two diffuse vertices \( x_{s-1} \), which precedes \( x_s \), and \( x_{s+1} \), which is generated via hemispherical sampling from
Supposing that \( x_s \) is on an ideal reflector, it is useful to define a few terms:

\[
\begin{align*}
\omega_i &= \hat{x}_{s-1} - x_s & \text{Direction backwards from } x_s \\
\omega_o &= \hat{x}_{s+1} - x_s & \text{Direction forwards from } x_s, \text{ reflection of } \omega_i \\
p_s &= \text{Potential path technique with } x_s \text{ and } x_{s+1} \text{ as endpoints} \\
p_{s-1} &= \text{Potential path technique with } x_s \text{ and } x_{s-1} \text{ as endpoints} \\
p_{s+1} &= \text{Potential path technique with } x_{s+1} \text{ and } x_{s+2} \text{ as endpoints}.
\end{align*}
\]

Suppose technique \( p_{s+1} \) was actually used to generate the complete path. We know that in this case \( \omega_o \) and \( \omega_i \) satisfy the delta functions in the BRDF and PDF at \( x_s \), since hemispherical sampling guarantees this by construction, and we can ignore the deltas for the whole path PDF.

Once we begin to consider technique \( p_s \), we run into the problem that there is a specular endpoint to one of the subpaths. Despite the fact that \( x_{s+1} \) was generated along reflection direction \( \omega_o \), there is a zero probability that that point could have been selected by any sampling method from one subpath to match the specular reflection at \( x_s \) on the other subpath. As such, in this case the complete path probability density for technique \( p_{s-1} \) is 0. This case is easy to detect by using a specular flag, and setting the PDF to 0 whenever any subpath endpoint is flagged.

Suppose that we continue stepping in this direction, computing complete path PDFs for \( p_{s-1}, p_{s-2}, \) and \( p_{s-3} \), using the relative PDF technique in section 4.2.4. The specular flag sets the PDFs \( p_{s-1} \) and \( p_{s-2} \) to 0 since \( x_s \) is a subpath endpoint. The problem now is that when using the relative path probabilities, this loses all information for computing the PDF \( p_{s-3} \), which has non-specular endpoints and quite possibly non-zero throughput. This is the reason to keep non-specular components recorded, since while the delta functions do not cancel in evaluations with specular subpath endpoints, they do in other cases past them. In practice, what this amounts to is using the relative path PDF technique with only the (computable, well-defined) non-specular PDF components, but "reporting" zero values to the weighting heuristic whenever a specular flag is encountered on a subpath endpoint.
Table A.1: Combinations of specular and diffuse components in cameras.

<table>
<thead>
<tr>
<th>Positional</th>
<th>Directional</th>
<th>Example Camera</th>
<th>Analogous Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffuse</td>
<td>Diffuse</td>
<td>Finite-aperture camera</td>
<td>Diffuse area light</td>
</tr>
<tr>
<td>Diffuse</td>
<td>Specular</td>
<td>Parallel-projection camera</td>
<td>Beam light (area laser)</td>
</tr>
<tr>
<td>Specular</td>
<td>Diffuse</td>
<td>Pinhole camera</td>
<td>Point light</td>
</tr>
<tr>
<td>Specular</td>
<td>Specular</td>
<td>Radiance probe</td>
<td>Point laser</td>
</tr>
</tbody>
</table>

A.4 Camera Sampling

Like surfaces and emitters, cameras can also have diffuse and specular components for their surfaces and viewing directions. Another way to think about this is to consider cameras to be importance emitters, in which case it is easy to describe what different combinations of components will do. Table A.1 gives some examples of cameras with various positional and directional component types. A camera with a specular position has a shape with at least one infinitesimal dimension, such as a 0-dimensional point for a pinhole camera, and a camera with a specular direction similarly has at least one missing dimension in the directions it selects, such as only "seeing" along a single direction or in a single plane.

A.4.1 Pinhole Camera

Generally, sampling a pinhole camera consists of two steps: selecting a point on the image plane, and creating a ray from that point through the pinhole. Taken together, these are simply a way of choosing only relevant directions for the diffuse directional component of the pinhole camera. When computing path probabilities, it is necessary to find the probability density of selecting each vertex of the path with respect to area, but pinhole cameras create two difficulties in doing this.

The first is that selection of a point on the camera is defined entirely by a delta function,

\[ p_A(x) = \delta(x - x_0), \]

where \( x_0 \) is the position of the pinhole. The solution here is the same as previously discussed with specular surfaces and emitters; the delta function is replaced with a flag so that it is only considered non-zero in cases where it is known beforehand that its
condition is satisfied. Specifically, when dealing with a path that begins at a pinhole camera, we know ahead of time that the selection of the point on the camera succeeds, but when considering the probability of potentially hitting the camera with a random ray, the delta function equals zero.

The second problem with using a pinhole camera is that finding the PDF \( p_A(x_1) \) of the first point hit by the camera ray is not entirely straightforward. The solution here is to perform several PDF conversions, starting with the area PDF of the point on the image plane, going through a solid angle PDF at the pinhole, and converting back into an area PDF at \( x_1 \).

Suppose that tracing begins with a particular pixel somewhere on the image plane (as is illustrated in 4.6). Regardless of size of the image plane, if a point in that pixel is selected uniformly, its probability density is

\[
p_{A}(x_{\text{pixel}}) = \frac{1}{A_{\text{pixel}}}. \tag{A.43}
\]

Once the image plane is set to a particular size, its the distance between \( x_{\text{pixel}} \) and the pinhole has some value \( r \). Connecting \( x_{\text{pixel}} \) to the pinhole also allows us to find \( \theta_{\text{pixel}} \), the angle between the image plane normal and the connecting direction \( \omega \) to the pixel (again, refer to figure 4.6). Using these values, and equation A.24 to find the PDF with respect to solid angle of selecting \( \omega \),

\[
p_{\sigma}(\omega) = p_{A}(x_{\text{pixel}}) \frac{r^2}{A_{\text{pixel}} \cos \theta_{\text{pixel}}} = \frac{r^2}{A_{\text{pixel}} \cos \theta_{\text{pixel}}} \cos \theta_{\text{pixel}}. \tag{A.44}
\]

Note that scaling the image plane while maintaining the same field of view does not affect this value. Scaling the plane by \( s \) also scales its distance to the pinhole by \( s \), and simple similar-triangles reasoning means that \( r \) is also scaled by \( s \), so while the pixel area is scaled by \( s^2 \), \( r^2 \) increases proportionately. The camera ray is defined as \( (x_0, -\omega) \), and using equation A.24 again allows us to find the area PDF of the point hit by this ray,

\[
p_{A}(x_1) = p_{\sigma}(-\omega) \frac{|N_{x_1} \cdot -\omega|}{||x_1 - x_0||^2} = \frac{r^2}{A_{\text{pixel}} \cos \theta_{\text{pixel}} \ ||x_1 - x_0||^2}. \tag{A.45}
\]

Intuitively, this equation should make sense. The first term is simply a directional PDF defined by entirely by the image plane position \( x_{\text{pixel}} \), allowing us to limit the sampling domain of directions so they only hit the image plane. The second term captures the effects of distance and foreshortening on the surface patch hit by the camera ray.
Bibliography


