Spring 6-1-2022

A bidirectional formulation for Walk on Spheres

Yang Qi
Yang.Qi.GR@Dartmouth.edu

A bidirectional formulation for Walk on Spheres

A Thesis
Submitted to the Faculty
in partial fulfillment of the requirements for the
degree of

Master of Science

in

Computer Science

by Yang Qi

Guarini School of Graduate and Advanced Studies
Dartmouth College
Hanover, New Hampshire

May 2022

Examining Committee:

Wojciech Jarosz

Bo Zhu

Peter Winkler

F. Jon Kull, Ph.D.
Dean of the Guarini School of Graduate and Advanced Studies

Abstract

Poisson's equations and Laplace's equations are important linear partial differential equations (PDEs) widely used in many applications. Conventional methods for solving PDEs numerically often need to discretize the space first, making them less efficient for complex shapes. The random walk on spheres method (WoS) is a grid-free Monte-Carlo method for solving PDEs that does not need to discrete the space. We draw analogies between WoS and classical rendering algorithms, and find that the WoS algorithm is conceptually identical to forward path tracing.

We show that solving the Poisson's equation is equivalent to solving the Green's function for every pair of points in the domain. Inspired by similar approaches in rendering, we propose a novel WoS reformulation that operates in the reverse direction. Besides this, using the corrector function enables us to use control variates to estimate the Green's function. Implementations of this algorithm show improvement over classical WoS in solving Poisson's equation with sparse sources. Our approach opens exciting avenues for future algorithms for PDE estimation which, analogous to light transport, connect WoS walks starting from sensors and sources and combine different strategies for robust solution algorithms in all cases.

Acknowledgment

First and foremost, I would like to express my deepest appreciation to my research supervisor, professor Wojciech Jarosz for guiding me in the research and teaching the wonderful course rendering algorithms. His patience in answering everything and his knowledge have always been very helpful throughout my research and learning experiences. I am grateful to learn from him and conduct reach under his supervision.

I'm extremely grateful to Dario Seyb and Benedikt Bitterli for valuable suggestions and help during the writing process. Chatting with them helped me a lot when I was stuck on research. The completion of this thesis would not have been possible without their help.

Thanks also to Lorie Loeb and Bo Zhu for giving the lectures on all aspects of computer graphics. I also would like to thank all my friends for giving me great times in New Hampshire. Lastly, I am grateful to my parents and my girlfriend Yimeng Shang for their generous support all these years.

Chapter 5, Chapter 6, and Chapter 8 of this thesis are based on a paper submission to EGSR which I have co-authored with Dario Seyb, Benedikt Bitterli and Wojciech Jarosz.

Contents

Abstract ii

Acknowledgment iii

Contents iv

1 Introduction 1
- 1.1 Related Work . . . 2
- 1.2 Thesis Overview . . . 3

2 Partial Differential Equations 4
- 2.1 Two Linear PDEs . . . 4
- 2.2 The Green's Method . . . 5
- 2.3 Stochastic Differential Equation . . . 7

3 Rendering and Monte-Carlo Methods 9
- 3.1 Monte-Carlo Methods . . . 9
  - 3.1.1 Importance Sampling . . . 9
- 3.2 Rendering . . . 10
  - 3.2.1 The Path Tracing Algorithm . . . 10
  - 3.2.2 The Three-Point Form of the Transport Equations . . . 11
  - 3.2.3 The Path Integral Formulation . . . 12
  - 3.2.4 Multiple Importance Sampling . . . 13

4 Random Walks on Spheres 14
- 4.1 Integral Equation of Poisson's Equations . . . 14
- 4.2 WoS Algorithm . . . 14

5 Bidirectional formula for Green's Functions 16
- 5.1 Integral Equations of Green's functions . . . 16
- 5.2 Source and sensor expansions. . . . 18
- 5.3 Analogies to light transport, path spaces, and MIS. . . . 19

6 Reverse WoS Algorithms 21
- 6.1 Estimating the Green's Function . . . 21
- 6.2 Solving the Poisson's Equation . . . 22
  - 6.2.1 Solving the Source Solution v . . . 22
  - 6.2.2 Solving the Boundary Solution w . . . 23
- 6.3 A Two-Pass Algorithm . . . 24
- 6.4 Bias Compensation . . . 24
- 6.5 Combining Forward and Reverse Walks . . . 25

7 Control variates 28
- 7.1 The Corrector Function . . . 28

7.2 Solving the Poisson's Equation 29

8 Implementation and Results **30**
8.1 Algorithm implementation 30
8.2 Comparison between forward and reverse WoS 30
8.2.1 Sparse boundary values 30
8.2.2 Sparse sources 32

9 Conclusion **33**
9.1 Limitations and Future Work. 33

Appendix **35**

A Pseudo-Code **36**

Bibliography **37**

1 Introduction

Partial differential equations(PDEs) can describe many physical phenomena and are used in many applications in the computer science area. For example, in fluid simulation, researchers are solving the Naiver-Stokes equation. In geometry processing, we can use Poisson's equation to reconstruct surfaces from 3D scanned data [1].

[1]: Kazhdan et al. (2006), 'Poisson surface reconstruction'

Unfortunately, almost all PDEs are not possible to solve analytically, this thesis will only focus on the theory and numerical methods for linear PDEs. For some simple linear PDEs, a lot of research has been done to let us write out the analytical solution using Fourier transformation or Green's functions [2]. However, these theories only enable us to calculate the analytical form of simple domains like a sphere or a plane.

[2]: Evans (2010), *Partial differential equations*

In real-world problems, the domains we want to solve PDEs are usually very complex. Thus mesh-based methods like the finite element method have been introduced to solve PDEs numerically for real-world use. These mesh-based PDEs solvers can handle arbitrary domains by splitting the domain into small parts and solving PDEs in those small parts. However mesh-based methods are inefficient to use when domains are extremely complex because they need to discretize the space first.

Another type of equation is the integral equation, which describes the relationship between a function and an integral. Some PDEs and integral equations are interchangeable because we can perform an integral operation on both sides of a PDE or perform a derivative operation to change between them. The rendering equation [3] is an integral equation which is the governing equation describing how the light interacts with the geometry or media in the world. Methods for solving the rendering equation start from mesh-based methods such as radiosity [4, 5] while Monte-Carlo methods were thought as too slow for solving the equation. However, in the past decade, the movie industry has shifted to use a Monte-Carlo method called the path-tracing algorithm to solve the equation [6, 7]. Monte-Carlo method has been proved to be stable and efficient in complex scenes compared to mesh-based methods.

[3]: Kajiya (1986), 'The Rendering Equation'

[4]: Cohen et al. (1986), 'An Efficient Radiosity Approach for Realistic Image Synthesis'

[5]: Cohen et al. (1993), *Radiosity and Realistic Image Synthesis*

[6]: Christensen et al. (2016), 'The Path to Path-Traced Movies'

[7]: Fascione et al. (2017), 'Path Tracing in Production (Parts 1 and 2)'

The random walk on spheres (WoS) method [8], introduced to

[8]: Muller (1956), 'Some Continuous Monte Carlo Methods for the Dirichlet Problem'

graphics by [9] recently is a Monte-Carlo method for solving linear PDEs. It uses the connection between parabolic PDEs and stochastic processes (Feynmann-Kac's theorem) and uses the local analytical solution to solve PDEs. Both the WoS algorithm and the path-tracing algorithm can be viewed as solving a linear recursive integral equation. Since WoS is also a Monte-Carlo method, it shares the same advantage as the path-tracing algorithm. WoS does not need to discrete the space, making it suitable for handling complex geometry domains. Comparisons between the WoS method and mesh-based methods have been well-explored, we refer to Sawhney and Crane [9] and Sawhney et al.[10] for a thorough discussion. The similarity in theory and the algorithm between WoS and path-tracing inspired us to generalize some techniques from rendering to WoS. In this thesis, we will develop new methods based on the WoS framework.

[9]: Sawhney et al. (2020), 'Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-based Methods on Volumetric Domains'

[10]: Sawhney et al. (2022), 'Grid-Free Monte Carlo for PDEs with Spatially Varying Coefficients'

1.1 Related Work

A major development in rendering is the invention of "Two-Pass" methods, Initially developed for radiosity [4, 5, 11] this idea has been proved to be successful in Monte-Carlo methods in the form of photon-mapping [12, 13] and VPLs/many-light rendering [14–19] and more recently, the methods [20–27] for rending volumetric media. These "Two-Pass" methods will first perform a "backward" pass to simulate light transport from light sources and store the information for retrieving illumination in some data structures. After that, they use a "forward" pass to gather the illumination for sensor points. These methods can connect paths from sensor points to source points more easily compared to a purely "forward" path-tracing algorithm, thus, making the estimator more robust in general.

Our main contribution is a new set of WoS algorithms that can start random walks from source points and distribute energy into the domain along the walk. We can directly rasterize these walks or look up into these walks after in a second "forward" sensor path, mimicking a form of "final gather" [11] to reduce structured sampling artifacts. Besides this, we purpose another way to use the WoS algorithm to solve the PDEs using the control variates. For now, although our algorithm can only use some heuristics to choose from how many forward steps we want to take. Eventually, we hope we can combine all strategies using multiple importance

[4]: Cohen et al. (1986), 'An Efficient Radiosity Approach for Realistic Image Synthesis'

[5]: Cohen et al. (1993), *Radiosity and Realistic Image Synthesis*

[11]: Reichert (1992), 'A Two-Pass Radiosity Method Driven by Lights and Viewer Position'

[12]: Jensen (2001), *Realistic Image Synthesis Using Photon Mapping*

[13]: Jensen (1996), 'Global Illumination Using Photon Maps'

[14]: Dachsbacher et al. (2014), 'Scalable Realistic Rendering with Many-Light Methods'

[15]: Keller (1997), 'Instant Radiosity'

[16]: Walter et al. (2005), 'Lightcuts: A Scalable Approach to Illumination'

[17]: Walter et al. (2006), 'Multidimensional Lightcuts'

[18]: Walter et al. (2012), 'Bidirectional Lightcuts'

[19]: Hašan et al. (2007), 'Matrix Row-Column Sampling for the Many-Light Problem'

[20]: Jensen et al. (1998), 'Efficient Simulation of Light Transport in Scenes with Participating Media Using Photon Maps'

[21]: Jarosz et al. (2008), 'The Beam Radiance Estimate for Volumetric Photon Mapping'

[22]: Jarosz et al. (2011), 'A Comprehensive Theory of Volumetric Radiance Estimation Using Photon Points and Beams'

[23]: Jarosz et al. (2011), 'Progressive Photon Beams'

[24]: Bitterli et al. (2017), 'Beyond Points and Beams'

[25]: Deng et al. (2019), 'Photon Surfaces for Robust, Unbiased Volumetric Density Estimation'

[26]: Novák et al. (2012), 'Virtual Ray Lights for Rendering Scenes with Participating Media'

[27]: Novák et al. (2012), 'Progressive Virtual Beam Lights'

sampling [28] and can make the algorithm fully bidirectional [29, 30], making the Monte-Carlo PDEs solver robust enough for use in different areas.

[28]: Veach et al. (1995), 'Optimally Combining Sampling Techniques for Monte Carlo Rendering'

[29]: Georgiev et al. (2012), 'Light Transport Simulation with Vertex Connection and Merging'

[30]: Hachisuka et al. (2012), 'A Path Space Extension for Robust Light Transport Simulation'

1.2 Thesis Overview

This thesis consists of nine chapters. In Chapter 2 we will discuss the background knowledge of partial differential equations, which will go over the content from the traditional viewpoint and the probabilistic viewpoint. In Chapter 3 we will discuss the Monte-Carlo method and how it is used to solve the problem in rendering. After that, we will describe the classical WoS method in Chapter 4. Next, in Chapter 5 and Chapter 6 we will derive the integral equation for Green's functions and how to use this equation to solve PDEs. Chapter 7 will introduce another method for solving PDEs using the WoS algorithm. We show the results and comparison between our methods and the classical WoS method in Chapter 8

Partial Differential Equations | 2

A partial differential equation(PDE) is an equation describing the property of the partial derivatives of a multi-variable function $u(x)$. In this thesis, we will use $U \in \mathbb{R}^n$ to denote the domain of the function we are solving and ∂U to denote the boundary of the U. In this chapter we will first discuss the Laplace's equation and the Poisson's equation, then describe two approaches for solving them, one is using the Green's function and another is using the probabilistic meaning of PDEs.

2.1 Two Linear PDEs

The Laplace's Operator

First we will introduce the Laplace's operator: Δ. In n-dimensional Euclidean space $\mathbb{E}^n$ with the natural Cartesian coordinates $\{x_i : i = 1, 2, \cdots, n\}$, then the Laplace's operator can be written as the sum of all the unmixed second order partial derivatives, $\Delta = \sum_{i=1}^{n} \frac{\partial^2}{\partial {x_i}^2}$. If we apply the Laplace's operator to a function u, the result Δu is called the Laplacian of u. Using the definition of the Laplace's operator under the natural Cartesian coordinates, the Laplacian of u at x is $\Delta u(x) = \sum_{i=1}^{n} \frac{\partial^2 u(x)}{\partial {x_i}^2}$.

An important property of the Laplace operator is that it is linear, which means given two functions v, w and two real numbers a, b we have:

$$\Delta(v + w) = \Delta v + \Delta w \tag{2.1}$$

Having defined the Laplace's operator, we can now introduce two partial differential equations we will mainly focus on, the Laplace's equation and the Poisson's equation.

The Laplace's Equation

Given a Dirichlet boundary condition g, which is a function defined on ∂U, describing the value of u on the boundary. The

Laplace's equation of u on U can be written as:

$$\begin{aligned} \Delta u(x) &= 0 && \text{if } x \in U \\ u(x) &= g(x) && \text{if } x \in \partial U. \end{aligned} \tag{2.2}$$

The Laplace's equation can be viewed as describing the heat diffusion in the steady state. $\Delta u(x) = 0$ states that there are no additional heat sources or sinks in the system. In this case, the heat at position $u(x)$ is entirely defined by the heat on the boundary $g(x)$.

The Poisson's equation

The Poisson's equation is a generalization of the Laplace's equation where additional heat sources or sinks are allowed. This means the Laplacian of $u(x)$ is not 0 everywhere but equals to a source term $f(x)$ defined in U. The Poisson's equation can be written as:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{if } x \in U \\ u(x) &= g(x) && \text{if } x \in \partial U. \end{aligned} \tag{2.3}$$

Both Laplace's equation and Poisson's equation are linear PDEs, which means the unknown function u and it derivatives are all linear.

2.2 The Green's Method

The Green's function

A general way to solve a linear PDE is called Green's method [2]. To apply this approach, let's first consider a special Poisson's equation where it's source term is a delta function δ_y and the boundary value is 0:

[2]: Evans (2010), *Partial differential equations*

$$\begin{aligned} \Delta\, u(x) &= \delta_y(x) && \text{if } x \in U \\ u(x) &= 0 && \text{if } x \in \partial U. \end{aligned} \tag{2.4}$$

We will denote the solution of this Poisson's equation by $\mathcal{G}_y^U(x)$. A common nice property for linear PDEs is that solution is symmetric, ($\mathcal{G}_y^U(x) = \mathcal{G}_x^U(y)$), to show this symmetric property we will rewrite it as $\mathcal{G}(x \leftrightarrow y)$ and omit the superscript when it is the entire domain U. Also, we want to expand the definition of

the Green's function's definition to $\mathbb{R}^n$, so we define the Green's function to be 0 if x or y are not in U. The expansion still follows the symmetric property and is still the solution to Equation 2.4.

Then let us consider a Poisson's equation with an arbitrary source function f and boundary value 0.

$$\begin{aligned} \Delta u(x) &= f(x) \quad \text{if } x \in U \\ u(x) &= 0 \quad\;\; \text{if } x \in \partial U \end{aligned} \tag{2.5}$$

Intuitively, the Green's function can be thought of as the amount of energy transported from y to x inside U. Then, if we add up all the sources inside the domain, it should be the solution to Equation 2.5. This is true because of the linear property of the operator, if we fix the point x and convolve the Green's function $\boldsymbol{G}(x \leftrightarrow *)$ with the source term $f(*)$ over the domain $v(x) = \int_U f(y)\, \boldsymbol{G}(x \leftrightarrow y)\, \mathrm{d}y$ will give us a solution to Equation 2.5 [2].

[2]: Evans (2010), *Partial differential equations*

$$u(x) = \int_U f(y)\, \boldsymbol{G}(x \leftrightarrow y)\, \mathrm{d}y. \tag{2.6}$$

Poisson's Kernel.

We have discussed how to solve the Poisson's equations only have source term Equation 2.5, the boundary value are handled through the *Poisson's kernel* $\boldsymbol{P}(x \to z) = \boldsymbol{P}(z \leftarrow x)$. Where $x \in U$ and $z \in \partial U$ and the arrow will always points to the point on the boundary. $\boldsymbol{P}(x \to z)$ is defined to be the normal derivative of the Green's function. Let $n(z)$ be the local normal of ∂U at z, we have:

$$\boldsymbol{P}(x \to z) = \boldsymbol{P}(z \leftarrow x) \coloneqq \frac{\partial \boldsymbol{G}(x \leftrightarrow z)}{\partial n(z)}, \tag{2.7}$$

We will follow a similar notation as for Green's functions. $\boldsymbol{P}(x \to z)$ will denote the Poisson's kernel for U, and $\mathcal{P}(x \to z)$ will denote the Poisson's kernel of a subdomain.

Representation Formula.

Given the Green's function and its Poisson kernel, the solution for general Poisson equations can be expressed using the representation formula [2]

[2]: Evans (2010), *Partial differential equations*

$$u(x) = \int_U f(y)\, \boldsymbol{G}(x \leftrightarrow y)\, \mathrm{d}y + \int_{\partial U} g(z)\, \boldsymbol{P}(x \to z)\, \mathrm{d}z, \tag{2.8}$$

2.3 Stochastic Differential Equation

In this section, we will discuss Poisson's equation in a probabilistic way. The Poisson's equation can be viewed as describing heat diffusion in steady state. Heat can be thought of as the energy on the particles and if the particle is dense in the domain, the path they move will be a Brownian-walk. This suggests that the Poisson's equation have a close relationship with Brownian motion. In the rest of this section, we will introduce some basic definitions of the stochastic process and introduce Feynman-Kac's theorem, which is the fundamental equation that shows the relationship between stochastic processes and PDEs.

Brownian Walk

Given a point x, the Brownian walk, or the Wiener process X starts at x is a stochastic process that maps time t to a random variable $X(t)$, which has the following properties:

- Starting point: $X(0) = x$
- Independent increments: for every t, $X(t+u) - X(t)$ are independent of the past values $X(s), s < t$
- Gaussian increments: $X(t+u) - X(t)$ follows the normal distribution $\mathcal{N}(0, u)$
- Continuity: $X(t)$ is continuous in t

Stopping Time

Suppose we have a Brownian walk X starting at x, we define the stopping time of X on U, T^U to be the first time X hits ∂U:

$$T^U = \inf\{t : X(t) \in \partial U\}. \tag{2.9}$$

$X(T^U)$ is a random variable, because a the Brownian walk can hit any point z on ∂U with some probability density function.

Feynman-Kac's Theorem

Feynman-Kac's theorem [31] shows a close relationship between stochastic processes and PDEs. It suggests that the solution of the Laplace's equation (2.2) can be expressed as:

[31]: Øksendal (2003), *Stochastic Differential Equations*

$$u(x) = \mathbb{E}\left[g(X(T^U))\right]. \tag{2.10}$$

This means that the solution of the Laplace's equation at a point x is equal to the expectation of the boundary value at where the Brownian walk X hits the boundary. If there are source terms inside the scene, the Feynman-Kac's theorem says besides the boundary value, we also need to take into account all the sources along the Brownian walk path, the solution to (2.3) can be written as:

$$u(x) = \mathbb{E}\left[g(X(T^U)) + \int_0^{T^U} f(X(t))\,\mathrm{d}t\right]. \tag{2.11}$$

One important property of Poisson's kernel can be found by comparing the integral of the boundary term in the representation formula (2.8) and Feynman-Kac's formula with no source (2.10). Since they are handling the same boundary term, the probability density of $X(T^U)$ should be equal to the Poisson kernel of U, $p^{\partial U}(z) = \boldsymbol{\mathcal{P}}^U(x, z)$. Thus, the Poisson kernel must satisfy the property of being a probability density function, which is the integral of the Poisson kernel is 1 over the boundary.

$$\int_{\partial U} \boldsymbol{\mathcal{P}}(x \to z)\,\mathrm{d}z = 1 \tag{2.12}$$

Rendering and Monte-Carlo Methods

3

In this chapter, we review the Monte-Carlo method for estimating an integral and how it is used for solving the rendering equations.

3.1 Monte-Carlo Methods

The underlying concept of Monte-Carlo methods is to use randomness to solve a deterministic problem. A typical example is using Monte-Carlo to calculate an integration numerically. Let Ω be the domain of a positive function f, suppose we want to solve the following integral:

$$I = \int_{\Omega} f(x)\,\mathrm{d}x. \tag{3.1}$$

If we sample $x \in \Omega$ according to the probability distribution $p(x)$. Then we can estimate the integral (Equation 3.1) by:

$$\langle I \rangle = \frac{f(x)}{p(x)}. \tag{3.2}$$

This is because the expectation of the right hand side of Equation 3.2 is:

$$\mathbb{E}\left[\frac{f(x)}{p(x)}\right] = \int_{\Omega} \frac{f(x)}{p(x)} p(x)\,\mathrm{d}x = \int_{\Omega} f(x)\,\mathrm{d}x. \tag{3.3}$$

If we take multiple samples $x_1, x_2, \cdots, x_n$, the law of large number ensures us the average of these estimates will converge to the real integral value:

$$\lim_{n \to \inf} \sum_{i=1}^{n} \frac{f(x)}{p(x)} = I. \tag{3.4}$$

3.1.1 Importance Sampling

Different choices of probability density $p(x)$ used in sampling will lead to different variances for the estimation of the integral value. A good choice of $p(x)$ is to let $p(x)$ be proportional to the function f. This sampling method is called *importance sampling*, it optimizes the variance when estimating the integral. However, in most cases, it is hard to sample x proportional to the function we

are integrating. For example, if we want to calculate the integral of the product of two arbitrary functions f and g:

$$I = \int_{\Omega} f(x)\, g(x)\, \mathrm{d}x. \tag{3.5}$$

In this case, an efficient way to sample x is sampling x proportional to f or g [32].

[32]: Veach et al. (1995), 'Bidirectional Estimators for Light Transport'

3.2 Rendering

Physically-based rendering aims to simulate light transport as in the real world. The governing integral equation that describes the light transport is the rendering equation [3]. Suppose we want to calculate $L_o(p, \omega_o)$, which is the radiance along output direction ω_o at location p it will equals to the emission along ω_o, $L_e(p, \omega_o)$ at p plus the indirect light, which equals to the integral of the bidirectional scattering distribution function(BSDF) f times the incoming light L_i over the upper hemisphere. The rendering equation is written as:

[3]: Kajiya (1986), 'The Rendering Equation'

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{S^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| \, \mathrm{d}\omega_i \tag{3.6}$$

$$L_i(p, \omega) = L_o(t(p, \omega), -\omega), \tag{3.7}$$

where $t(p, \omega)$ is the ray-casting function that gives the first intersection point of the ray staring at p in the direction ω.

3.2.1 The Path Tracing Algorithm

The rendering equation is an integral equation, thus we can use Monte-Carlo method to solve it. Suppose we want to calculate the radiance $L_o(p, \omega_o)$, we first evaluate the emission light $L_e(p, \omega_o)$ at the current location. Then, we randomly sample a new direction ω_i according to a probability density function $p(\omega_i)$ and use a single-sample Monte-Carlo estimator to estimate the integral term.

$$\langle L_o(p, \omega_o) \rangle = L_e(p, \omega_o) + \frac{f(p, \omega_o, \omega_i) L_i(p, \omega_i)}{p(\omega_i)}. \tag{3.8}$$

Then we can use the second equation $L_i(p, \omega) = L_o(t(p, \omega), -\omega)$ to replace $L_i(p, \omega_i)$ on the right hand side, which will give us:

$$\langle L_o(p, \omega_o) \rangle = L_e(p, \omega_o) + \frac{f(p, \omega_o, \omega_i) L_o(t(p, \omega_i) - \omega_i)}{p(\omega_i)}. \quad (3.9)$$

To evaluate the $L_o(t(p, \omega_i), -\omega_i)$, we can do the same step again. The iteration process can be terminated by the Russian roulette method. The path tracing algorithm follows these steps we just discussed to solve the rendering equation, which generates the ray path in one direction. There is also plenty of "backward" methods such as light tracing, photon mapping [33], and virtual point lights (VPLs) [15] and bidirectional path tracing [32] that generate paths in both directional and combine them.

[33]: Jensen et al. (1995), 'Photon Maps in Bidirectional Monte Carlo Ray Tracing of Complex Objects'

[15]: Keller (1997), 'Instant Radiosity'

[32]: Veach et al. (1995), 'Bidirectional Estimators for Light Transport'

3.2.2 The Three-Point Form of the Transport Equations

The ray-casting function $t(p, \omega)$ and the recursive L_i on the right hand side in Equation 3.6 makes this equation very complex. In the following of this chapter, we will rewrite the rendering equation into the form of $L = \int_\Omega f_j(x) \, \mathrm{d}\mu$. First, we will use a geometry term function to change the light transport equation into the surface area form. Let $V(x \leftrightarrow x')$ be the visibility function:

$$\begin{aligned} V(x \leftrightarrow x') &= 0 \quad \text{if } x' \text{ and } x \text{ are mutually visible} \\ V(x \leftrightarrow x') &= 1 \quad \text{else.} \end{aligned} \quad (3.10)$$

Then we define the geometry term function $G(x \leftrightarrow x')$:

$$G(x \leftrightarrow x') = V(x \leftrightarrow x') \frac{|\cos(\theta) \cos(\theta')|}{\|x - x'\|^2}. \quad (3.11)$$

G can be viewed as the change of measure between a point measure dx to the surface area measure $dA(x)$. Then we also want to remove the directional variables ω_i, ω_o. Thus, we define

$$L_o(x \to x') = L(x, \omega), \quad (3.12)$$

where $\omega = \frac{x' - x}{\|x' - x\|}$ is the unit-length direction pointing from x to x'. Next, we can also write the BSDF as:

$$f(p, \omega_o, \omega_i) = f(x \to x' \to x''), \quad (3.13)$$

where $\omega_i = \frac{x-x'}{\|x-x'\|}$ and $\omega_o = \frac{x''-x'}{\|x''-x'\|}$ are two unit-length directions.
Then Equation 3.6 can be written in the three-point form:

$$L_o(x' \to x'') = L_e(x' \to x'') + \int_M f(x \to x' \to x'') L_i(x \to x') G(x \leftrightarrow x') \, \mathrm{d}A(x) \tag{3.14}$$

3.2.3 The Path Integral Formulation

We want to write the rendering equation in the form of doing an integral of a single space and removing the recursion part. To do this, we need to define the path space and the measure of the path space. Let Ω_k denote all possible paths of length k.

$$\Omega_k = \{\bar{x} : \bar{x} = x_0 x_1 \cdots x_k\}. \tag{3.15}$$

Then, to write out an integral over Ω_k, let's define a new measure μ_k over Ω_k which is equal to the product measure of $A(x_i)$. Let $D \in \Omega_k$ be a set of paths of length k, $\mu(D)$ is given by:

$$\mu(D) = \int_D \mathrm{d}A(x_0) A(x_1) \cdots A(x_k). \tag{3.16}$$

Now let us generalize this measure to the path space Ω, which is the set of all possible paths:

$$\Omega = \cup_{k=1}^{\infty} \Omega_k. \tag{3.17}$$

Let $D \in \Omega$, the natural generalization of measure μ_k is the sum of the measures of the paths of each length:

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

After defining the measure over the path space, we can recursively expand Equation 3.14 to rewrite it into an integral over the path space:

$$L_o(x_0 \to x_1) = \sum_{k=1}^{\infty} \int_{M^{k+1}} L_e(x_0 \to x_1) G(x_0 \leftrightarrow x_1) \prod_{i=1}^{k-1} f(x_{i-1} \to x_i \to x_{i+1}) G(x_i \leftrightarrow x_{i+1}) \, \mathrm{d}A(x_0) A(x_1) \cdots A(x_k). \tag{3.19}$$

The integrand can be viewed as a function f_j of a path $\bar{x}$, we can simplify the equation to be:

$$L_o(x_0 \to x_1) = \int_\Omega f_j(\bar{x}) \, \mathrm{d}\mu(\bar{x}). \tag{3.20}$$

3.2.4 Multiple Importance Sampling

Writing the rendering equation in the path integral way makes it possible to perform multiple importance sampling on it. Suppose we want to estimate $L = \int_\Omega f(x)\,\mathrm{d}\mu$. If we have two strategies of sampling x with probability density $p_1(x)$ and $p_2(x)$. Given two samples x, y sampled from these two strategies respectively, we can combine two strategies to estimate the integral:

$$\langle L \rangle = \frac{1}{p_1(x) + p_2(x)} f(x) + \frac{1}{p_1(y) + p_2(y)} f(y). \tag{3.21}$$

In practice, when using multiple importance sampling for rendering, we have two strategies for sampling a path i.e. simulating the path from the sensor points or simulating the path from the light sources. Using multiple importance sampling, we can combine these two strategies and produce a robust Monte-Carlo estimator [34].

[34]: Veach (1997), 'Robust Monte Carlo Methods for Light Transport Simulation'

4 Random Walks on Spheres

In this chapter, we will discuss the random walk on spheres(WoS) method. The WoS algorithm is a Monte-Carlo method for solving Poisson's equations numerically. Unlike traditional methods, WoS does not need to discretize the space before solving PDEs, which makes it efficient for handling PDEs in complex geometry. Similar to rendering, the WoS method can be viewed as solving an iterative integral equation.

4.1 Integral Equation of Poisson's Equations

This section will discuss the integral equation of Poisson's equations we are solving. Suppose u is the solution to the Poisson's equation Equation 2.3 and let B_x be ball centered at x. The representation theorem Equation 2.8 can be generalized to works on B_x, making it to be an *integral equation* for the solution:

$$u(x) = \underbrace{\int_{B_x} f(y)\, \mathcal{G}(x \to y)\, \mathrm{d}y}_{\text{volume term}} + \underbrace{\int_{\partial B_x} \overbrace{u(z)}^{=g(z) \text{ when } z \in \partial U}\, \mathcal{P}(x \to z)\, \mathrm{d}z}_{\text{boundary term}}. \quad (4.1)$$

This equation holds for any shape, meaning we can change B_x to be any shape S that contains x. In this thesis, we always use ball to be the shape where we do the calculation because the Green's function of a ball has an elegant analytical form. Also, B_x will always be the largest ball contained in U which is not necessary but can reduce the iteration steps, making the algorithm more efficient.

4.2 WoS Algorithm

After writing the solution u in the form an integral equation. We can use Monte-Carlo methods to solve it in a recursive way. At the point x_i, we sample a $y \sim p_{B_{x_i}}(y)$ inside ball B_{x_i}, sample

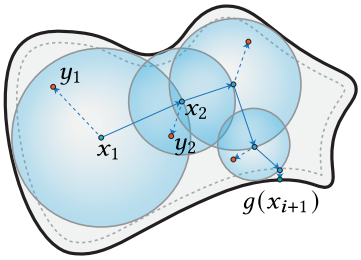


Figure 4.1: A WoS path with single source point sample y_i at each vertex.

$x_{i+1} \sim p_{\partial B_{x_i}}(x_{i+1})$ on the boundary of the ball, and evaluate

$$\langle u(x_i) \rangle = \frac{f(y)\, \mathcal{G}(x_i \to y)}{p_{B_{x_i}}(y)} + \frac{\langle u(x_{i+1}) \rangle\, \mathcal{P}(x_i \to x_{i+1})}{p_{\partial B_{x_i}}(x_{i+1})}. \tag{4.2}$$

This is a single-sample estimator of solution u for both the recursive part and the local part, which we denote $\langle u \rangle$. Notably, $\langle u(x_{i+1}) \rangle$ appears on the right-hand side, requiring a recursive evaluation of Equation 4.2: For each sample x_{i+1}, we select a new ball $B_{x_{i+1}}$ centered on x_{i+1} and recurse. When the point x_{i+1} we sampled is on the boundary, we can use the boundary value of the Poisson's equation $g(x_{i+1})$ to estimate the solution $u(x_{i+1})$. However, the probability of sampling a point exactly on the boundary is very small. In practice, this recursion process continues until we generate a sample x_{i+1} sufficiently close to the boundary ∂U of the domain (when the distance to the boundary at that point is less than ϵ), at which point we evaluate $g(x_{i+1})$ instead of recursing. Fig. 4.1 shows an example WoS path with one source sample y_i inside each ball B_x.

Bidirectional formula for Green's Functions

5

The WoS algorithm needs to choose the next step randomly on a sphere, thus it has no little control of where random walks hit the boundary. Although we can perform importance sampling on the local source terms and the local Poisson's kernel in Equation 4.2, it is still a local optimize decision depending on the ball we used. Suppose the scene only has sparse (or even delta) sources or boundary values, the WoS algorithm cannot adjust its sampling strategy according to it.

A similar issue also appears in rendering, where it might be hard for a forward path tracing hit the light source. Using some "reverse" methods like photon mapping can solve this issue. In the following, we propose a "reverse" random walk on spheres algorithm, which starts WoS path from source points and boundary, thus it can perform importance sampling of the source term globally. We will follow the same idea behind Sec. 2.2, i.e. by considering the method for computing the Green's function first and then using it as the base for computing the solution to Equation 2.3. As we can see from Equation 2.8, if we have a way to estimate the Green's function for any two given points x, y, we can perform importance sampling on the source term and the boundary term. This sampling strategy will give a better global sampling choice in some scenes.

In this chapter, we will discuss the integral equation of the Green's function, which will be the cornerstone of the algorithm in the next chapter.

5.1 Integral Equations of Green's functions

Like in the WoS algorithm, to compute the Green's function of the domain we first want to have an integral equation for Green's function. There are many ways to derive the integral equation of Green's functions, we will use Feynman-Kac's theorem to derive it. First, we apply Feynman-Kac's theorem(Equation 2.11) to the PDE (Equation 2.4) to derive a probabilistic definition of the Green's function. Recall the Green's function $\boldsymbol{G}(x \leftrightarrow y)$ is defined to be the solution to (Equation 2.4). The source term of this Poisson's equation is the Dirac delta function at y, $\delta_y(X(t))$ and there are

no boundary values. Applying the Feynman-Kac's theorem gives us:

$$\mathcal{G}(x \leftrightarrow y) = \mathbb{E}\left[\int_0^{T^U} \delta_y(X(t))\,\mathrm{d}t\right]. \tag{5.1}$$

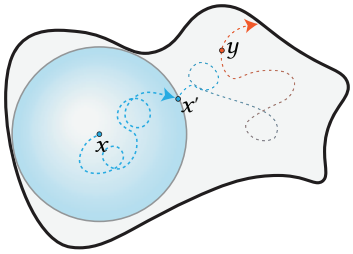


Figure 5.1: The Brownian walk X can be splitted into two parts.

As showed in Fig. 5.1, let's put a ball B_x centered at x such that B_x is contained in U and let T^U be the stopping time of a Brownian walk X on U. Because of the continuity of the Brownian walk, the Brownian walk must touch the ball B_x first before it continues the walk and hit the ∂U. Let T^{B_x} be the stopping time of X on B_x, then we can split the integral in Equation 5.1 into two parts.The first one is the time that the walk is inside B_x, $[0,\ T^{B_x}]$. The second part is the time after the walk hits ∂B_x, and until it hits the boundary of the domain $[T^{B_x},\ T^U]$:

$$\begin{aligned}\mathcal{G}(x \leftrightarrow y) &= \mathbb{E}\left[\int_0^{T^{B_x}} \delta_y(X(t))\,\mathrm{d}t + \int_{T^{B_x}}^{T^U} \delta_y(X(t))\,\mathrm{d}t\right] \\ &= \mathbb{E}\left[\int_0^{T^{B_x}} \delta_y(X(t))\,\mathrm{d}t\right] + \mathbb{E}\left[\int_{T^{B_x}}^{T^U} \delta_y(X(t))\,\mathrm{d}t\right].\end{aligned} \tag{5.2}$$

Notice that the first expectation above is the same as in Equation 5.1 by changing the domain to B_x, so it is equal to the Green's function on B_x. Because Brownian motion has independent increments, the integral in the second expectation can be viewed as doing a new Brownian walk starting at $x' = X(T^{B_x})$. Since the probability density of $X(T^{B_x})$ is equal to the Poisson kernel of the ball $\mathcal{P}(x \to *)$, let $X'(t) = X(t + T^{B_x})$ denotes the part of the Brownian walk after T^{B_x}, we can rewrite the second expectation as:

$$\begin{aligned}&\mathbb{E}\left[\int_{T^{B_x}}^{T^U} \delta_y(X(t))\,\mathrm{d}t\right] \\ &= \int_{\partial B_x} \Pr[X(T^{B_x}) = x']\,\mathbb{E}\left[\int_0^{T^U - T^{B_x}} \delta_y(X'(t))\,\mathrm{d}t\right]\mathrm{d}y \\ &= \int_{\partial B_x} \mathcal{P}(x \to x')\mathcal{G}(x' \leftrightarrow y)\,\mathrm{d}x'.\end{aligned} \tag{5.3}$$

Combining Equation 5.2 and Equation 5.3, we derived the mean value theorem of the Green's function:

$$\mathcal{G}(x \leftrightarrow y) = \mathcal{G}(x \to y) + \int_{\partial B_x} \mathcal{P}(x \to x')\mathcal{G}(x' \leftrightarrow y)\,\mathrm{d}x'. \tag{5.4}$$

Thanks to the symmetry of the Green's function, if we swap the x, y and do the same step on y over ball B_y we can get the mean value theorem for y:

$$\mathcal{G}(x \leftrightarrow y) = \mathcal{G}(x \leftarrow y) + \int_{\partial B_y} \mathcal{G}(x \leftrightarrow y')\mathcal{P}(y' \leftarrow y)\, \mathrm{d}y'. \quad (5.5)$$

5.2 Source and sensor expansions.

The two mean value theorems for x, y enables us to choose which direction to expand the recursive integral. We can either expand it in the "forward" direction (by changing x using Equation 5.4) or in the "reverse" direction (by changing y using Equation 5.5). We can expand the recursive integral in the forward direction two times and we will have:

$$\begin{aligned}\mathcal{G}(x_0 \leftrightarrow y_0) = & \underbrace{\mathcal{G}(x_0 \rightarrow y_0)}_{\text{1 "bounce" transport}} \\ & + \underbrace{\int_{\partial B_{x_0}} \mathcal{P}(x_0 \rightarrow x_1)\mathcal{G}(x_1 \rightarrow y_0)\, \mathrm{d}x_1}_{\text{2 "bounce" transport}} \\ & + \underbrace{\iint_{\partial B_{x_0} \times \partial B_{x_1}} \mathcal{P}(x_0 \rightarrow x_1)\mathcal{P}(x_1 \rightarrow x_2)\mathcal{G}(x_2 \leftrightarrow y_0)\, \mathrm{d}x_2 x_1}_{\text{3+ "bounce" transport}}.\end{aligned} \quad (5.6)$$

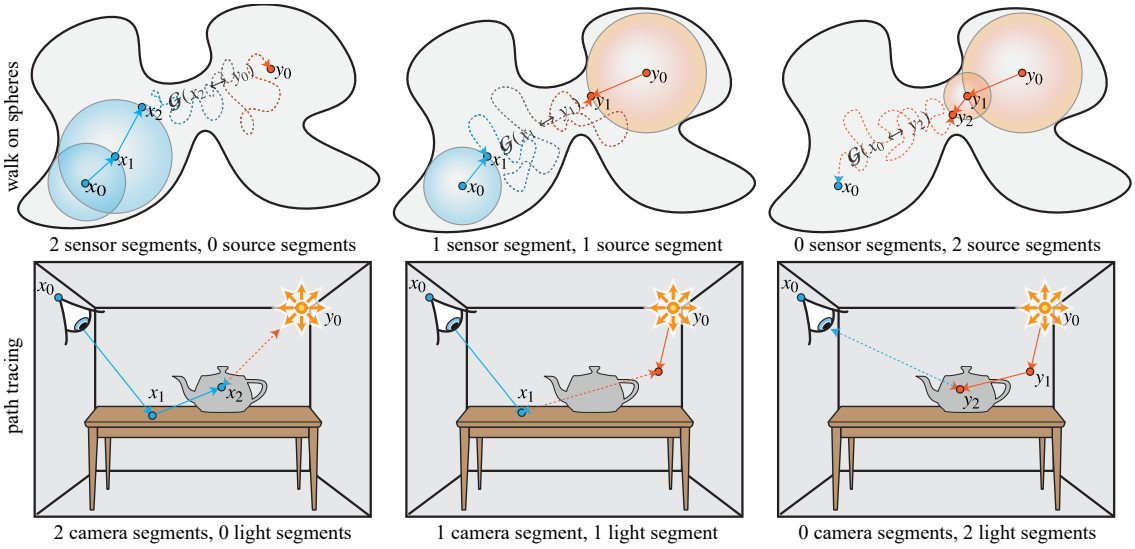


Figure 5.2: Top row: We show the different ways our new formalism lets us construct WoS paths, corresponding to Equation 5.6, Equation 5.8 and Equation 5.7 (left to right respectively). Bottom row: These methods correspond closely to forward path tracing, bidirectional path tracing and light tracing (left to right respectively).

Alternatively, if we change the direction we expand, by doing two "reverse" expansion from y_0, we can have:

$$\begin{aligned}\boldsymbol{\mathcal{G}}(x_0 \leftrightarrow y_0) = & \underbrace{\mathcal{G}(x_0 \leftarrow y_0)}_{\text{1 "bounce" transport}} \\ & + \underbrace{\int_{\partial B_{y_0}} \mathcal{G}(x_0 \leftarrow y_1)\mathcal{P}(y_1 \leftarrow y_0)\,\mathrm{d}y_1}_{\text{2 "bounce" transport}} \\ & + \underbrace{\iint_{\partial B_{y_0}\times\partial B_{y_1}} \boldsymbol{\mathcal{G}}(x_0 \leftrightarrow y_2)\mathcal{P}(y_2 \leftarrow y_1)\mathcal{P}(y_1 \leftarrow y_0)\,\mathrm{d}y_2 y_1\,.}_{\text{3+ "bounce" transport}}\end{aligned} \tag{5.7}$$

Finally, because we can freely choose from Equation 5.4 or Equation 5.5 in each step. Thus we can expand the integral equation from both direction once to get a "bidirectiona" expansion, here we show a "bidirectional" expansion by expanding first with Equation 5.4 followed by Equation 5.5:

$$\begin{aligned}\boldsymbol{\mathcal{G}}(x_0 \leftrightarrow y_0) = & \underbrace{\mathcal{G}(x_0 \rightarrow y_0)}_{\text{1 "bounce" transport}} \\ & + \underbrace{\int_{\partial B_{x_0}} \mathcal{P}(x_0 \rightarrow x_1)\mathcal{G}(x_1 \leftarrow y_0)\,\mathrm{d}x_1}_{\text{2 "bounce" transport}} \\ & + \underbrace{\iint_{\partial B_{x_0}\times\partial B_{y_0}} \mathcal{P}(x_0 \rightarrow x_1)\boldsymbol{\mathcal{G}}(x_1 \leftrightarrow y_1)\mathcal{P}(y_1 \leftarrow y_0)\,\mathrm{d}y_1 x_1\,.}_{\text{3+ "bounce" transport}}\end{aligned} \tag{5.8}$$

These expansion choice now gives us different ways to estimate the Green's function $\boldsymbol{\mathcal{G}}(x_0 \leftrightarrow y_0)$.

5.3 Analogies to light transport, path spaces, and MIS.

We illustrate the three example expansions from Equations 5.6–5.8 in Fig. 5.2 (bottom). Since both the Green's function and the rendering equation are recursive Fredholm integrals, there is a natural analogy between these equations and different strategies of bidirectional light transport Fig. 5.2 (top). In fact, repeatedly expanding the recursion in the three-point form of the rendering

equation is the typical process to obtain Veach's path integral formulation [34], which provides methods like bidirectional path tracing [32, 35] with a powerful way to combine all these strategies into one algorithm using MIS [28].

[34]: Veach (1997), 'Robust Monte Carlo Methods for Light Transport Simulation'
[32]: Veach et al. (1995), 'Bidirectional Estimators for Light Transport'
[35]: Lafortune et al. (1993), 'Bi-Directional Path Tracing'
[28]: Veach et al. (1995), 'Optimally Combining Sampling Techniques for Monte Carlo Rendering'

Unfortunately, the analogies depicted in Fig. 5.2 are imperfect. Although the Green's function itself is symmetric, the subdomain we used to perform the expansion in Equations 5.4 and 5.5 are different (a sphere ∂B_x or ∂B_y). This means that each distinct sequence of expansion directions results in a different shape of the integral domain making the path space different. This is easy to confirm by observing that the domains of integration for 3+ "bounce" transport in Equations 5.6–5.8 are all distinct. This is in contrast to the rendering equation, where all "bounces" happens on the geometry in the space, and using surface area measure ensures that expansion from either direction produces the exact same path space.

This means that while MIS can still be performed *within* a single path space choice, it is not immediately clear how to MIS *across* these different path space choices. Doing so would be akin to using MIS to combine VPLs, photon mapping, and bidirectional path tracing within a unified path space [29, 30]. Nevertheless, even before being cast into a unified path space, photon mapping and VPL methods proved highly successful using hand-crafted criteria for determining how many steps to take along a camera subpath before connecting to a light subpath. In the next sections we explore several such possible bidirectional combinations, and leave the exciting prospect of a unified path space for future work.

[29]: Georgiev et al. (2012), 'Light Transport Simulation with Vertex Connection and Merging'
[30]: Hachisuka et al. (2012), 'A Path Space Extension for Robust Light Transport Simulation'

Reverse WoS Algorithms 6

In the previous discussion, we have shown two integral equations of the Green's function and the expansion of the integral equations by choosing different direction to expand in each steps. In this chapter, we will discuss how to use these formulas to solve the Poisson equation numerically.

6.1 Estimating the Green's Function

We will first focus on how to estimate the Green's function using Monte-Carlo methods through the integral equations we discussed in the previous chapter. Through Equations 5.4 and 5.5, a one-sample Monte Carlo estimation of these will give us the forward and backward estimators of the Green's function, which can be written as:

$$\begin{aligned} \langle \boldsymbol{\mathcal{G}}(x \leftrightarrow y) \rangle &= \mathcal{G}(x \to y) + \frac{\mathcal{P}(x \to x') \langle \boldsymbol{\mathcal{G}}(x' \leftrightarrow y) \rangle}{p^{\partial B_x}(x')} \\ \langle \boldsymbol{\mathcal{G}}(x \leftrightarrow y) \rangle &= 0 \quad \text{if } x \in \partial U \text{ or } y \in \partial U, \end{aligned} \tag{6.1}$$

$$\begin{aligned} \langle \boldsymbol{\mathcal{G}}(x \leftrightarrow y) \rangle &= \mathcal{G}(x \leftarrow y) + \frac{\mathcal{P}(y \to y') \langle \boldsymbol{\mathcal{G}}(x \leftrightarrow y') \rangle}{p^{\partial B_y}(y')} \\ \langle \boldsymbol{\mathcal{G}}(x \leftrightarrow y) \rangle &= 0 \quad \text{if } x \in \partial U \text{ or } y \in \partial U, \end{aligned} \tag{6.2}$$

where x' and y' are points sampled on the boundary of the ball B_x and B_y, with densities $p^{\partial B_x}(x')$ and $p^{\partial B_y}(y')$. Equation 6.1 and Equation 6.2 show estimator of the Green's function in recursive form. At each recursive step, we can choose freely from Equation 6.1 or Equation 6.2 to generate next vertex x' or y' respectively. The recursion is stopped when the next vertex sampled land on the boundary ∂U where we know the Green's function is 0 there. However, the probability density of generating a vertex exactly on the boundary is 0, we will follow the same approach in forward WoS Chapter 4, using a small epsilon band as the boundary. In this case, we will estimate the Green's function by 0 if the distance from either x or y to the boundary is less than epsilon. Fig. 6.1 and Fig. 6.2 shows a purely forward and a purely reverse walk.

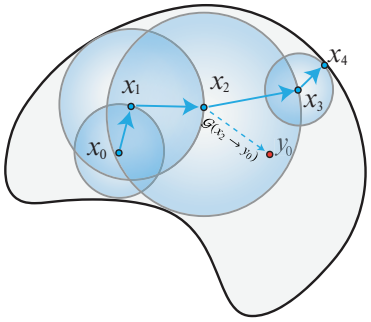


Figure 6.1: Following the forward walk $\{x_i\}$, we can estimate: $\langle \boldsymbol{\mathcal{G}}(x_0 \leftrightarrow y_0) \rangle = \mathcal{G}(x_2 \to y_0)$

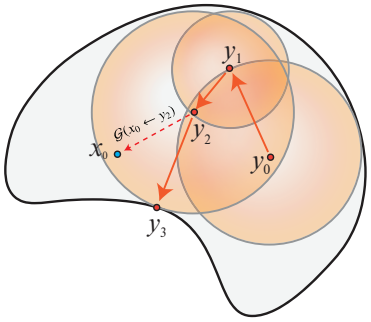


Figure 6.2: Following the reverse walk $\{y_i\}$ we can estimate: $\langle \boldsymbol{\mathcal{G}}(x_0 \leftrightarrow y_0) \rangle = \mathcal{G}(x_0 \leftarrow y_2)$

6.2 Solving the Poisson's Equation

After having an estimator of the Green's function, now we can start discussing how to solve the Poisson's equation. To begin with, we first split the Poisson's equation Equation 2.3 into two PDEs, the source-only part v and the boundary-only part w, which satisfy

$$\begin{aligned} \Delta v(x) &= f(x) & \Delta w(x) &= 0 & &\text{if } x \in U \\ v(x) &= 0 & w(x) &= g(x) & &\text{if } x \in \partial U \end{aligned} \tag{6.3}$$

From the linearity of the Laplace operator, we can see that the solution $u(x) = v(x) + w(x)$ satisfies the original Poisson problem (Equation 2.3), with $\Delta(v + w) = \Delta v + \Delta w = f$ and $v + w = g$. Because solutions to the Poisson's equation are unique [2], allowing us to retrieve the original solution u exactly via the sub-problems v and w.

6.2.1 Solving the Source Solution v

We begin with using the representation formula Equation 2.8 to the split Poisson's equation for v (Equation 6.3). Since the boundary value is 0, we obtain:

$$v(x) = \int_U f(y)\boldsymbol{G}(x \leftrightarrow y)\,\mathrm{d}y. \tag{6.4}$$

Calculating this integral by a one-sample Monte-Carlo method can be done in two steps. First, we sample one point y according to density $p^U(y)$. Then we we use the estimator described in Sec. 6.1 to estimate $\boldsymbol{G}(x \leftrightarrow y)$ by doing a WoS path with either Equation 6.1 or Equation 6.2 each step until the path hits the boundary.

$$\langle v(x) \rangle = \frac{f(y)\langle \boldsymbol{G}(x \leftrightarrow y) \rangle}{p^U(y)}. \tag{6.5}$$

By reducing the Poisson problem to estimating the Green's function, we get great flexibility in estimating v in any combination of forward and backward steps. Notably, unlike the classical forward WoS algorithm, Equation 6.5 allows us to sample y proportional to the source term $f(y)$ over the whole domain.

6.2.2 Solving the Boundary Solution w

Same as Sec. 6.2.1, we first apply representation formula for the boundary-only Poisson's equation of w (Equation 6.3), we obtain

$$w(x) = \int_{\partial U} \boldsymbol{\mathcal{P}}(x \to z) g(z) \, \mathrm{d}z. \tag{6.6}$$

Estimating this equation requires knowing the Poisson kernel $\boldsymbol{\mathcal{P}}(x \to z)$ over the whole domain. Luckily, since the Poisson kernel is the normal derivative of the Green's function we can reduce it to the estimating of the Green's function:

$$\begin{aligned} \boldsymbol{\mathcal{P}}(x \to z) &= \frac{\partial \boldsymbol{\mathcal{G}}}{\partial n}(z) && (6.7)\\ &= \lim_{\xi \to 0} \frac{\boldsymbol{\mathcal{G}}(x \leftrightarrow z + \xi n) - \boldsymbol{\mathcal{G}}(x \leftrightarrow z)}{\xi} && (6.8)\\ &= \lim_{\xi \to 0} \frac{\boldsymbol{\mathcal{G}}(x \leftrightarrow z + \xi n)}{\xi}, && (6.9) \end{aligned}$$

where the last step used the fact that $\boldsymbol{\mathcal{G}}(x \leftrightarrow z) = 0$ for $z \in \partial U$.

We can approximate Equation 6.9 at the cost of bias by choosing a finite ξ instead of taking the limit, which is equivalent to taking the finite differences of the Green's function (see Fig. 6.3). This allows us to estimate the Poisson function with the Green's function estimators introduced in Sec. 6.1:

$$\langle \boldsymbol{\mathcal{P}}(x \to z) \rangle = \frac{\langle \boldsymbol{\mathcal{G}}(x \leftrightarrow z + \xi n) \rangle}{\xi}. \tag{6.10}$$

By inserting Equation 6.10 into Equation 6.6 and applying Monte Carlo integration to z, we obtain the estimator

$$\langle w(x, z) \rangle = \frac{g(z) \langle \boldsymbol{\mathcal{P}}(x \to z) \rangle}{p^{\partial U}(z)} = \frac{g(z) \langle \boldsymbol{\mathcal{G}}(x \leftrightarrow z + \xi n) \rangle}{p^{\partial U}(z) \xi}. \tag{6.11}$$

for w, where z is sampled from density $p^{\partial U(z)}$ on the boundary ∂U (e.g. proportional to boundary term $g(z)$). Much like the estimator for v, we have great flexibility in estimating the Green's function using any combination of forward- and backward steps. Unlike the classical forward WoS algorithm, Equation 6.11 allows us to sample z proportional to the boundary term $g(y)$ over the entire boundary.

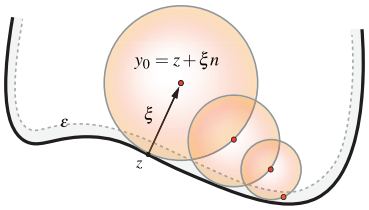


Figure 6.3: We sample z on the boundary of the domain and then push it away along the normal n to start the WoS path at $y_0 = z + \xi n$ to estimate $\boldsymbol{\mathcal{P}}(* \to z)$.

6.3 A Two-Pass Algorithm

In practice, we are usually interested in not only in estimating the solution $f(x)$ at a single point, but over a dense region. This allows for an efficient algorithm that *reuses* reverse walks.

If we recursively expand Equation 6.5 or Equation 6.11 using the reverse estimator Equation 6.2, then at each step y_i the walk contributes to *all* points x within the ball B_{y_i}. This is analogous to VPLs, where at each bounce the VPL contributes its flux to all points in the scene, modulated by a geometry term. The equivalent of the geometry term for reverse WoS is the Green's function on the ball, $\mathcal{G}(x \leftarrow y_i)$.

We exploit this by first performing a large number of reverse walks from the boundary and sources in the domain, and store the ball and Green's function estimate of each step of each walk in a spatial data structure. To estimate the solution at x, we then simply look up into the data structure to obtain all balls that overlap with x, and evaluate the Green's function estimate times $\mathcal{G}(x \leftarrow y_i)$. A pseudo code for this algotihm is showed in Alg. 1. This is analogous to two-pass many-light algorithms common in rendering [14]. We show an example of a solution estimate using increasing number of reverse WoS walks in Fig. 6.4.

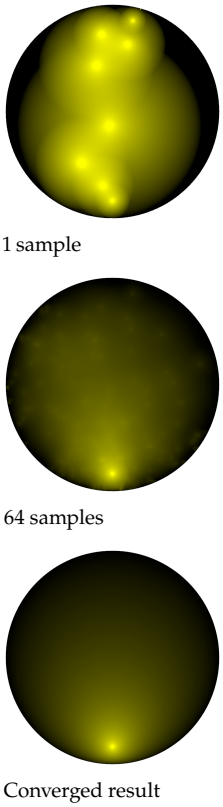


Figure 6.4: The solution estimated via reverse WoS with 1, 64, and 160,000 samples to a Poisson equation with a single point source inside a circle with a black boundary.

6.4 Bias Compensation

Although the finite difference method in Sec. 6.2.2 makes it possible to estimate the Poisson kernel via the Green's function, the finite step ξ introduces additional bias. This is made worse by the fact that $\xi >> \epsilon$ for practical reasons: If ξ is of smaller or equal magnitude to ϵ, then reverse walks starting at the boundary will immediately terminate; larger values of ξ are needed to "push off" walks away from the boundary.

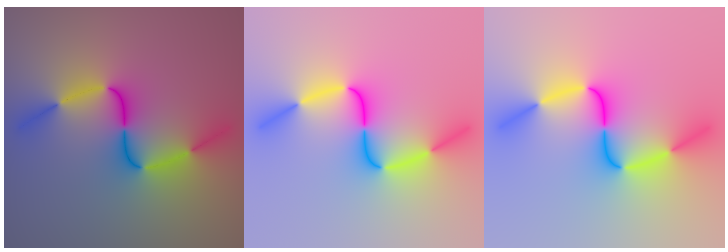


Figure 6.5: Left: Reverse WoS without normalization is darker than the ground truth. Mid: Reference image created by forward WoS. Right: Reverse WoS with normalization has the same brightness as the reference image.

In practice, the bias from finite differences manifests as darkening of the solution due to reverse walks terminating early (Fig. 6.5). This means that, unlike Equation 2.12, the finite difference Poisson kernel no longer integrates to 1.

We can compensate for this fact by *renormalizing* the Poisson kernel. While evaluating reverse walks, in addition to the solution w we also estimate the integral of the Poisson kernel at each point using a second Monte Carlo estimate,

$$\int_{\partial U} \langle \boldsymbol{\mathcal{P}}(x \to z) \rangle \, \mathrm{d}z \approx \frac{1}{M} \sum_{j=0}^{M} \frac{\langle \boldsymbol{\mathcal{P}}(x \to z_j) \rangle}{p^{\partial U}(z_j)} = \langle \mathcal{P}_{\text{norm}}(x) \rangle, \tag{6.12}$$

where $z_1, \ldots, z_M$ are the boundary samples generated in the course of solving for w.*

Dividing Equation 6.11 by the estimate of the normalization factor $\langle \mathcal{P}_{\text{norm}}(x) \rangle$ then allows us to compensate for the systematic darkening caused by the finite difference Poisson kernel:

$$\langle w(x, z) \rangle = \frac{g(z) \langle \boldsymbol{\mathcal{P}}(x \to z) \rangle}{p^{\partial U}(z) \langle \mathcal{P}_{\text{norm}}(x) \rangle}. \tag{6.13}$$

Although this estimate is still biased, the apparent error is much reduced (Fig. 6.5, right).

6.5 Combining Forward and Reverse Walks

Forward- and reverse walk on spheres share some of the same tradeoffs as forward and reverse transport simulation in rendering. Sparse, high frequency sources are much more difficult to find for forward methods than reverse methods; simultaneously, it is much more difficult to get even coverage of the sensor points for reverse methods than forward methods.

For example, if we are interested in computing the solution in only a small subset of a scene, it may be much more difficult for reverse walks to contribute to the solution. This is analogous to light tracing performing poorly when the camera only views a small part of a scene.

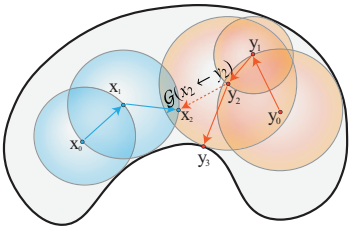
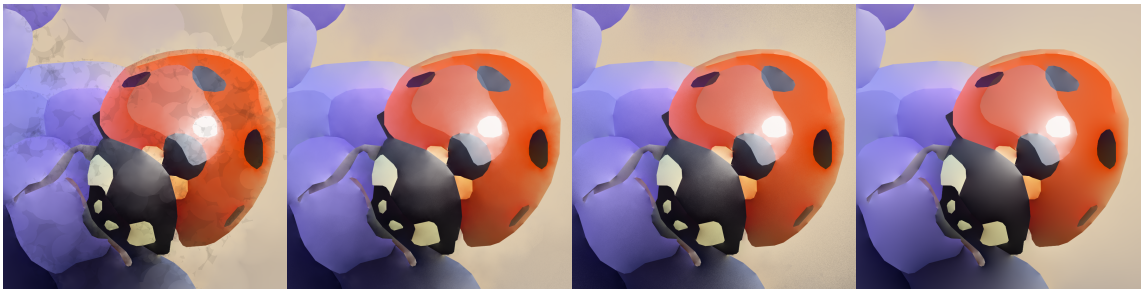


Figure 6.6: We can perform forward walks from sensor points and look up the solution from the reverse WoS's result. In this figure, we take two forward steps and then estimate the Green's function at x_2.

* Care should be taken that $p^{\partial U}(z) > 0$ over the entire boundary. Even if the boundary term $g(z) = 0$ for some of the boundary, the Poisson kernel is not.

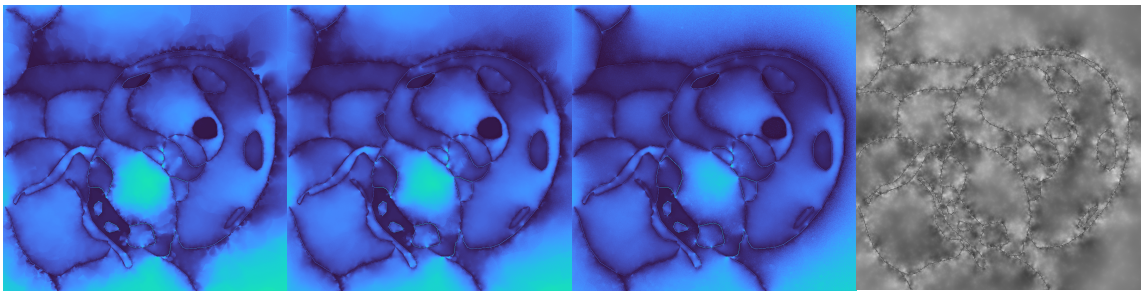


Purely reverse WoS
Runtime: 13.3s

Poisson's kernel heuristic
(T=0.9) Runtime: 17.4s

Poisson's kernel heuristic
(T=0.5) Runtime: 61.3s

Converged reference



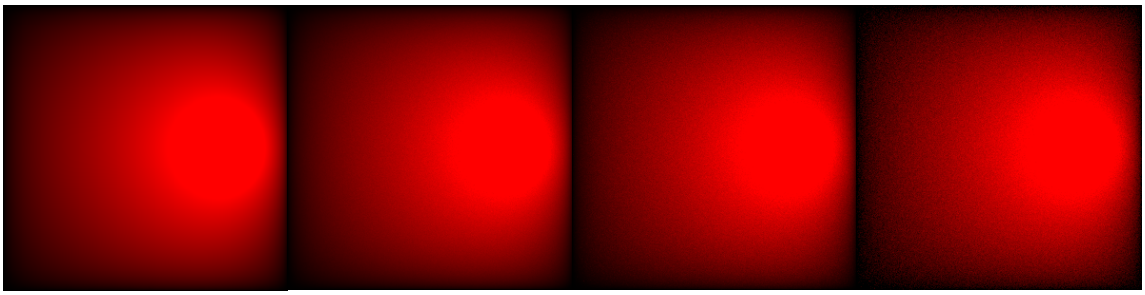
Mean variance: 5.19e-3

Mean variance: 4.35e-3

Mean variance: 3.60e-3

Normalization factor

Figure 6.7: Left three of the top row: results with different choice of WoS paths. Left three of the bottom row: variance per pixel of these choices, calculated by doing 160 independent runs. Bottom right: the normalization factor estimated by the reverse WoS. Insufficient samples in the reverse WoS will lead to structured artifacts in the region hard for a reverse path to covered (Top left). Using final gather with Poisson's kernel heuristic will make the algorithm do more samples in region where reverse WoS failed to have a good result and is able to fix the artifacts (Middle left). However, setting the threshold too strict will lead to poor performance (Middle right).



Reverse WoS
Runtime: 14.7s

Forward steps = 4
Runtime: 25.5s

Forward steps = 16
Runtime: 37.0s

Forward WoS
Runtime: 96.2s

Figure 6.8: Left: reverse WoS; middle two: bidirectional WoS with 4 and 16 forward final gather steps; right: forward WoS. We used a single point source in a square boundary, in this case, at each vertex, the forward WoS is solving the local mean value theorem analytically (this is not possible when there is a complex source term). We also tested final gather with different forward steps on many different scenes. However, from the results, we conclude that unless there is an efficient way to reuse forward samples or we are only interested in solving PDEs for a small portion of the scene, doing forward steps only introducing more noise and taking up more time for solving the source solution.

Although the different path spaces preclude robustly weighted combinations of all forward and reverse strategies using MIS (Sec. 5.3), we take inspiration from early light transport work [35] and choose between different combinations of forward and reverse strategies based on heuristics.

We already have access to a metric of how well reverse WoS performs: The integral of the finite difference Poisson kernel, $\langle \mathcal{P}_{\text{norm}}(x) \rangle$. If this estimate deviates significantly from 1, reverse WoS is performing poorly. This leads to a simple but effective heuristic for combining forward and reverse walks: Instead of computing the solution $f(x)$ from the data structure directly as in Sec. 6.3, we first evaluate $|\langle \mathcal{P}_{\text{norm}}(x) \rangle - 1| < T$ to see if the Poisson kernel norm deviates from unity by more than a threshold T. If it does, the reverse WoS solution is unreliable, and we perform one forward WoS step and repeat the procedure. This continues until the heuristic succeeds, at which point we look up into the data structure to estimate the solution (see Fig. 6.6). This is exactly analogous to final gather methods in graphics [11], and helps greatly to reduce artifacts (Fig. 6.7) at little extra cost (Fig. 6.8).

Control variates 7

In this chapter, we will discuss another way to estimate the Green's function, that is control variates. This method is estimating the Green's function by a function Φ minus the corrector function which equals $\Phi - \boldsymbol{\mathcal{G}}$. The corrector can be viewed as a solution to a Laplace's equation and Φ can be viewed as the Green's function in $\mathbb{R}^n$ with no boundary. In this thesis, we will introduce this idea through the mean value theorem of the Green's function Equation 5.4.

7.1 The Corrector Function

The proof for the mean value theorem Equation 5.4 does not require the subdomain to be a ball. We can use an arbitrary shape for it. Now suppose we find a large ball B that covers the entire domain, $U \subset B$. Let Φ denote the Green's function on this B, we can still apply the mean value theorem Equation 5.4 but using U as the subdomain, which will give us:

$$\Phi(x \leftrightarrow y) = \boldsymbol{\mathcal{G}}(x \leftrightarrow y) + \int_{\partial U} \boldsymbol{\mathcal{P}}(x \to x')\Phi(x' \leftrightarrow y)\, \mathrm{d}x'. \quad (7.1)$$

Move $\boldsymbol{\mathcal{G}}$ to left hand side and move Φ to right hand side:

$$\boldsymbol{\mathcal{G}}(x \leftrightarrow y) = \Phi(x \leftrightarrow y) - \int_{\partial U} \boldsymbol{\mathcal{P}}(x \to x')\Phi(x' \leftrightarrow y)\, \mathrm{d}x'. \quad (7.2)$$

Same as the mean value theorem of y Equation 5.5, we can swap x, y here too, which gives us:

$$\boldsymbol{\mathcal{G}}(x \leftrightarrow y) = \Phi(x \leftrightarrow y) - \int_{\partial U} \boldsymbol{\mathcal{P}}(y \to y')\Phi(y' \leftrightarrow x)\, \mathrm{d}y'. \quad (7.3)$$

In practice, it is natural to use the $B = \mathbb{R}^n$ since any domain will be covered by $\mathbb{R}^n$. Then Φ will be the fundamental solution of Poisson's equation on $\mathbb{R}^n$, which satisfy the following Poisson's equation with no boundary

$$\Delta_x \Phi(x \leftrightarrow y) = \delta_y(x), \quad (7.4)$$

where Δ_x means the Laplace's operator for x. Φ has a nice analytical form, let $A(n)$ denote the volumen of n dimension unit ball:

$$\Phi(x \leftrightarrow y) = \begin{cases} \frac{1}{2\pi} \log(|x-y|) & n = 2 \\ \frac{1}{n(n-2)A(n)} \frac{1}{|x-y|^{n-2}} & n \geq 3 \end{cases} \tag{7.5}$$

According to Equation 2.8, we can see the integral on the right hand side $\int_{\partial U} \boldsymbol{P}(x \to x')\Phi(x' \leftrightarrow y)\, \mathrm{d}x'$ is giving the solution to a Laplace's function:

$$\begin{aligned} \Delta u(x) &= 0 && \text{if } x \in U \\ u(x) &= \Phi(x \leftrightarrow y) && \text{if } x \in \partial U. \end{aligned} \tag{7.6}$$

The solution to this Laplace equationEquation 7.6 is called the corrector function of the Poisson/Laplace's equation [2].

7.2 Solving the Poisson's Equation

To estimate Green's function, we only need to solve the Laplace equation for corrector functions Equation 7.6. From Chapter 4 and Chapter 6, we can either use the classical WoS algorithm or the reverse WoS algorithm to solve the Laplace equation.

8 Implementation and Results

8.1 Algorithm implementation

Same as generating a forward walk, generating a reverse walk only requires querying the closest point to the boundary in order to expand the largest sphere. We implemented a 2D version of reverse WoS that is entirely on the CPU and use a standard acceleration structure [36] to make the closest point query efficient. In this case, the main bottleneck is drawing the Green's disks, which we currently do naively by testing all pixels within each disk's bounding box. Performing the reverse walk on the CPU but then splatting the Green's disks using rasterization on the GPU would likely result in a dramatic speedup. To implement our method one needs to evaluate $\mathcal{G}(x \to y)$ and $\mathcal{P}(x \to y)$. The concrete values of these depend on the PDE one is solving and we refer to the appendix in Sawhney et al. [10] for a comprehensive listing.

8.2 Comparison between forward and reverse WoS

We compared our method with forward WoS algorithm in multiple scenes to evaluate the efficiency, quality and robustness of our method. We found our method works better than the forward WoS in several cases because reverse WoS is able to importance sample the source term f and boundary value g in Equation 2.8 globally, making the algorithm focus more on the sources with high impact to the entire scene.

8.2.1 Sparse boundary values

Fig. 8.1 shows the estimated solution of a Laplace's equation of both algorithms using the same amount of time. In this example we set the boundary value to be 0 (black) at most boundary locations, leaving only a few small regions with colorful values along the circular boundary. A forward walk has no control of where the path will hit the boundary, so in a scene with sparse boundary

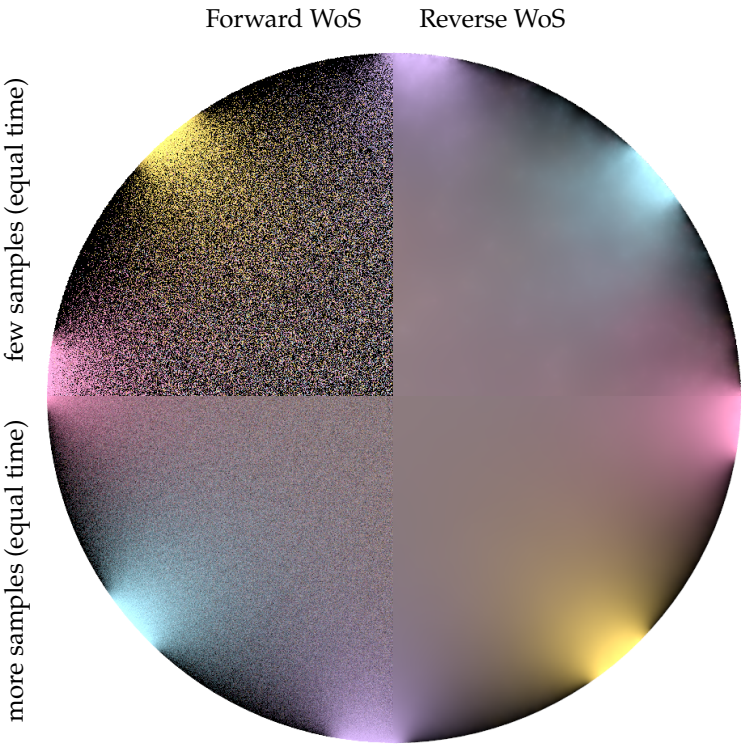


Figure 8.1: We compare forward WoS (left) to reverse WoS (right) at equal time with few samples (top) and with 16× as many samples (bottom). Reverse WoS produces a smooth result even with few samples, converging more quickly than forward WoS.

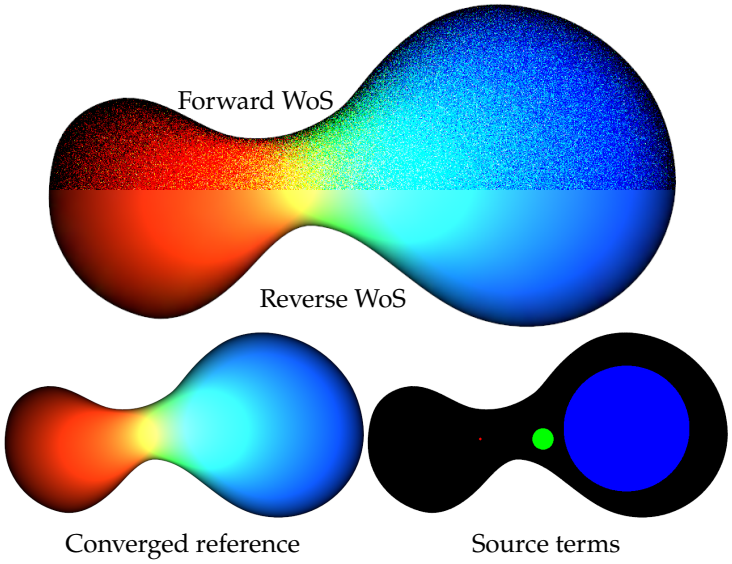


Figure 8.2: Here we perform an equal-time comparison of forward (top of split) vs. our reverse (bottom of split) WoS on a scene with three sparse sources (bottom right) and zero boundary conditions. Compared to the converged reference (bottom left), our approach with only 4e4 sampled paths produces visually better results than forward WoS with 3.24e6 total paths.

values, most walks are unlikely to receive a large contribution, resulting in high variance. Just as in purely unidirectional path tracing, the variance for forward WoS would get arbitrarily worse if we were to make the "lights" (boundary values) even more concentrated. In contrast, reverse WoS can importance sample the boundary values, dramatically reducing variance.

8.2.2 Sparse sources

Fig. 8.2 shows the estimated solution by reverse and forward WoS on a Poisson equation with black boundary, but three differently sized disk-shaped sources inside the domain. Since the source terms are spatially sparse, it is difficult for a forward WoS path to hit those disks sources and evaluate their contribution along the forward path. However, in the reverse WoS, since we know where the disks are located in space, we can easily importance sample f in Equation 2.8 when sampling the starting points of our paths. This importance sampling is essentially choosing global optimal sampled source points, while forward WoS can only sample local optimal choices at each step.

Conclusion 9

In this thesis, we presented the bidirectional formulation for Green's functions, taking the first step toward the bidirectional WoS algorithm. Using the bidirectional formulas described in Chapter 5, we derived and implemented WoS algorithms with different combinations of "forward" and "reverse" steps Chapter 6. Our algorithms surpass the traditional WoS method in multiple scenes Chapter 8.

9.1 Limitations and Future Work.

There are several topics that we do not address in this thesis. Particularly, the reverse WoS is only unbiased when estimating Green's function, the finite difference method we used to handle Poisson's kernelSec. 6.2.2 is not ideal. We hope there is an unbiased Monte-Carlo way to estimate the Poisson's kernel. On the theoretical side, we do not extend the set of equations that WoS can solve. First, the Feynmann-Kac theorem introduced in Sec. 2.3 can also give the solution to all parabolic partial differential equations[31]. For example, the WoS algorithm should also work on heat equations and other similar linear PDEs that include the time variable. Back to the rendering side, both the radical transport equation and the rendering equation are linear PDEs, which means our theory can also be generalized to those PDEs as long as we know the Green's function of a ball. Our method should have less variance compared with traditional path-tracing algorithms or even photon primitives because we can simulate an infinite number of bounces inside each ball using the Green's function.

Additionally, our method still requires the boundary condition to be Dirichlet boundary condition. While many real-world problems require other boundary conditions like Neumann and Robin boundary conditions. WoS algorithm could be more useful to those problems if it can handle different boundary conditions.

On the performance side, we only implemented a naive version of the algorithm. Different optimizations can be done to increase the performance. For example, the main bottleneck of the reverse WoS is rasterizing the ball, which could be solved easily if moved

onto a GPU. In our implementation, the next vertex is always uniformly sampled on the boundary of the ball to importance sample the Poisson's kernel of the ball, this might not be the best choice in all situations. When solving the boundary value part, because we always start walks close to the boundary, many walks will touch the boundary very quickly even before they contribute anything to the scene.

Related to the bidirectional WoS method, currently, our bidirectional algorithm described in Sec. 6.5 does not have a stable and nice heuristic for how to combine the forward and reverse walk(the integral of the Poisson's kernel only equals one if this is Poisson's equation). It may require doing multiple tests on different scenes to figure out when we should use a reverse step or a forward step in estimating Green's function. Finally, an obvious next step related to our contribution is to establish a single path integral formulation for all path construction strategies to allow for a robust combination of strategies via MIS as discussed in Sec. 5.3.

We hope Monte-Carlo methods can be used more in solving PDEs in various applications and hope our work gives a starting point to do so.

Appendix

Pseudo-Code A

Here we provide pseudo-code for the two-pass algorithm for solving the source solution $v(x)$ described in Sec. 6.3

Algorithm 1 A Two-pass Algorithm for Source Solution $v(x)$

Input: x
/ Sample N reverse WoS path and store them into L . */*
$L \leftarrow$ vertex_storage()
for $i \leftarrow 0$ to N **do**
  $y, p_y \leftarrow$ sample_source() *// Sample y according to pdf p_y*
  $s_y \leftarrow f(y)$ *// Evaluate the source term at y*
  $r_y \leftarrow$ distance_to_boundary(y_i)

  / Loop until the path hits the boundary. */*
  while $r_y > \epsilon$ **do**
    $r_y \leftarrow$ distance_to_boundary(y)
    L.store(y, p_y, s_y)
    $y \leftarrow$ sample_sphere_uniform(y, r_y) *// Continue the walk*
  end while
end for

/ Look up the solution $v(x)$ from stored reverse WoS vertices. */*
$v \leftarrow 0$
for (y, p_y, s_y) in L **do**
  if $|x - y| < r_y$ **then**
    $g \leftarrow \mathcal{G}(x \leftarrow y)$ *// Evaluate the local Green's function*
    $v \leftarrow v + g * s_y / p_y$
  end if
end for
return v

Bibliography

[1] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 'Poisson surface reconstruction'. In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Vol. 7. 2006 (cited on page 1).

[2] Lawrence C. Evans. *Partial differential equations*. Providence, R.I.: American Mathematical Society, 2010 (cited on pages 1, 5, 6, 22, 29).

[3] James T. Kajiya. 'The Rendering Equation'. In: *Computer Graphics (Proceedings of SIGGRAPH)* 20.4 (Aug. 1986), pp. 143–150. DOI: 10/cvf53j (cited on pages 1, 10).

[4] Michael F. Cohen et al. 'An Efficient Radiosity Approach for Realistic Image Synthesis'. In: *IEEE Computer Graphics & Applications* 6.2 (Mar. 1986), pp. 26–35. DOI: 10/c3mmt4 (cited on pages 1, 2).

[5] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. NY: Academic Press, 1993 (cited on pages 1, 2).

[6] Per H. Christensen and Wojciech Jarosz. 'The Path to Path-Traced Movies'. In: *Foundations and Trends® in Computer Graphics and Vision* 10.2 (Oct. 2016), pp. 103–175. DOI: 10/gfjwjc (cited on page 1).

[7] Luca Fascione et al. 'Path Tracing in Production (Parts 1 and 2)'. In: *ACM SIGGRAPH Courses*. Aug. 2017. DOI: 10/gfz2ck (cited on page 1).

[8] Mervin E. Muller. 'Some Continuous Monte Carlo Methods for the Dirichlet Problem'. In: *Annals of Mathematical Statistics* 27.3 (Sept. 1956), pp. 569–589. DOI: 10/cpxd3d (cited on page 1).

[9] Rohan Sawhney and Keenan Crane. 'Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-based Methods on Volumetric Domains'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39.4 (2020). DOI: 10.1145/3386569.3392374 (cited on page 2).

[10] Rohan Sawhney et al. 'Grid-Free Monte Carlo for PDEs with Spatially Varying Coefficients'. In: (Jan. 2022) (cited on pages 2, 30).

[11] Mark C. Reichert. 'A Two-Pass Radiosity Method Driven by Lights and Viewer Position'. M.Sc. Thesis. Ithaca, NY: Program of Computer Graphics, Cornell University, Jan. 1992 (cited on pages 2, 27).

[12] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. Natick, MA, USA: AK Peters, Ltd., 2001 (cited on page 2).

[13] Henrik Wann Jensen. 'Global Illumination Using Photon Maps'. In: *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*. Vienna: Springer-Verlag, June 1996, pp. 21–30. DOI: 10/fzc6t9 (cited on page 2).

[14] Carsten Dachsbacher et al. 'Scalable Realistic Rendering with Many-Light Methods'. In: *Computer Graphics Forum* 33.1 (Feb. 2014), pp. 88–104. DOI: 10/f5twgd (cited on pages 2, 24).

[15] Alexander Keller. 'Instant Radiosity'. In: *Annual Conference Series (Proceedings of SIGGRAPH)*. ACM Press, Aug. 1997, pp. 49–56. DOI: 10/fqch2z (cited on pages 2, 11).

[16] Bruce Walter et al. 'Lightcuts: A Scalable Approach to Illumination'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 24.3 (Aug. 2005), pp. 1098–1107. DOI: 10/dhp5d3 (cited on page 2).

[17] Bruce Walter et al. 'Multidimensional Lightcuts'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 25.3 (July 2006), pp. 1081–1088. DOI: 10/dzgsz7 (cited on page 2).

[18] Bruce Walter, Pramook Khungurn, and Kavita Bala. 'Bidirectional Lightcuts'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 31.4 (July 2012), 59:1–59:11. DOI: 10/gfzrcx (cited on page 2).

[19] Miloš Hašan, Fabio Pellacini, and Kavita Bala. 'Matrix Row-Column Sampling for the Many-Light Problem'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 26.3 (July 2007), 26:1–26:10. DOI: 10/djv68s (cited on page 2).

[20] Henrik Wann Jensen and Per H. Christensen. 'Efficient Simulation of Light Transport in Scenes with Participating Media Using Photon Maps'. In: *Annual Conference Series (Proceedings of SIGGRAPH)*. ACM Press, July 1998, pp. 311–320. DOI: 10/b64p36 (cited on page 2).

[21] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. 'The Beam Radiance Estimate for Volumetric Photon Mapping'. In: *Computer Graphics Forum (Proceedings of Eurographics)* 27.2 (Apr. 2008), pp. 557–566. DOI: 10/bjsfsx (cited on page 2).

[22] Wojciech Jarosz et al. 'A Comprehensive Theory of Volumetric Radiance Estimation Using Photon Points and Beams'. In: *ACM Transactions on Graphics* 30.1 (Jan. 2011), 5:1–5:19. DOI: 10/fcdh2f (cited on page 2).

[23] Wojciech Jarosz et al. 'Progressive Photon Beams'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 30.6 (Dec. 2011), 181:1–181:12. DOI: 10/fn5xzj (cited on page 2).

[24] Benedikt Bitterli and Wojciech Jarosz. 'Beyond Points and Beams: Higher-Dimensional Photon Samples for Volumetric Light Transport'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 36.4 (July 2017), 112:1–112:12. DOI: 10/gfznbr (cited on page 2).

[25] Xi Deng et al. 'Photon Surfaces for Robust, Unbiased Volumetric Density Estimation'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 38.4 (July 2019). DOI: 10.1145/3306346.3323041 (cited on page 2).

[26] Jan Novák et al. 'Virtual Ray Lights for Rendering Scenes with Participating Media'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 31.4 (July 2012), 60:1–60:11. DOI: 10/gbbwk2 (cited on page 2).

[27] Jan Novák et al. 'Progressive Virtual Beam Lights'. In: *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)* 31.4 (June 2012), pp. 1407–1413. DOI: 10/gfzndw (cited on page 2).

[28] Eric Veach and Leonidas J. Guibas. 'Optimally Combining Sampling Techniques for Monte Carlo Rendering'. In: *Annual Conference Series (Proceedings of SIGGRAPH)*. Vol. 29. ACM Press, Aug. 1995, pp. 419–428. DOI: 10/d7b6n4 (cited on pages 3, 20).

[29] Iliyan Georgiev et al. 'Light Transport Simulation with Vertex Connection and Merging'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 31.6 (Nov. 2012), 192:1–192:10. DOI: 10/gbb6q7 (cited on pages 3, 20).

[30] Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. 'A Path Space Extension for Robust Light Transport Simulation'. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 31.6 (Nov. 2012), 191:1–191:10. DOI: 10/gbb6n3 (cited on pages 3, 20).

[31] Bernt Øksendal. *Stochastic Differential Equations: An Introduction with Applications*. Sixth. Universitext. Berlin Heidelberg: Springer-Verlag, 2003 (cited on pages 8, 33).

[32] Eric Veach and Leonidas J. Guibas. 'Bidirectional Estimators for Light Transport'. In: *Photorealistic Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*. Springer-Verlag, 1995, pp. 145–167. DOI: 10/gfznbh (cited on pages 10, 11, 20).

[33] Henrik Wann Jensen and Niels Jørgen Christensen. 'Photon Maps in Bidirectional Monte Carlo Ray Tracing of Complex Objects'. In: *Computers & Graphics* 19.2 (Mar. 1995), pp. 215–224. DOI: 10/d9xr6q (cited on page 11).

[34] Eric Veach. 'Robust Monte Carlo Methods for Light Transport Simulation'. PhD thesis. Stanford University, Dec. 1997 (cited on pages 13, 20).

[35] Eric P. Lafortune and Yves D. Willems. 'Bi-Directional Path Tracing'. In: *Proceedings of the International Conference on Computational Graphics and Visualization Techniques (Compugraphics)*. Vol. 93. Alvor, Portugal, Dec. 1993, pp. 145–153 (cited on pages 20, 26).

[36] Rohan Sawhney et al. *fcpw*. https://github.com/rohan-sawhney/fcpw. 2021 (cited on page 30).