Monte Carlo Geometry Processing: A Grid-Free Approach to Solving Partial Differential Equations on Volumetric Domains

Rohan Sawhney

CMU-CS-24-125
May, 2024

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Keenan Crane (Chair)
Ioannis Gkioulekas
Matthew O'Toole
Gautam Iyer (Mathematical Sciences, CMU)
Matt Pharr (NVIDIA)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Copyright © 2024 Rohan Sawhney

This research was sponsored by National Science Foundation awards IIS1943123 and CCF1717320, the David and Lucille Packard Foundation award 201868047, Apple, Inc. award 1012669, OCULUS VR LLC award PO1075912, gifts from nTop and Disney, and an NVIDIA Graduate Fellowship. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: partial differential equations, Monte Carlo methods, geometry processing, physics-based rendering and simulation, computer graphics

Abstract

This thesis develops Monte Carlo algorithms based on the *walk on spheres (WoS)* method to reliably solve fundamental partial differential equations (PDEs) like the Poisson equation on geometrically complex domains. Elliptic PDEs are a basic building block of algorithms and applications throughout science, engineering, and geometric computing. Yet despite decades of research on methods for solving such PDEs, conventional solvers still struggle to deal with the level of geometric complexity found in the natural world. A constant challenge is the need for spatial discretization, which traditionally involves dividing the domain into a high-quality volumetric mesh or grid to perform PDE-based analysis. Unfortunately, this approach does not scale well to modern computer architectures as it is inherently sequential and memory intensive. It also falters when dealing with imperfect data containing poorly-shaped elements or self-intersections. These shortcomings together hinder the ability of scientists, engineers and designers to analyze geometric data and iterate on designs.

Walk on spheres makes a radical departure from conventional PDE solvers by reformulating the problem in terms of recursive integral equations that can be solved using the Monte Carlo method, allowing it to avoid volumetric mesh generation and function space approximation altogether. Furthermore, since these integral equations closely resemble those found in light transport theory, one can leverage deep knowledge from Monte Carlo rendering to build new algorithms for solving PDEs.

In this work, we take inspiration from rendering to generalize WoS to solve a much broader set of linear elliptic PDEs on solid regions of R^N. We develop complete "black box" solvers encompassing integration, variance reduction and acceleration. Our solvers share many benefits with Monte Carlo methods from rendering: no volumetric meshing, trivial parallelism, output-sensitive evaluation of the PDE solution and its gradient without the need to solve a globally-coupled system of equations, and the ability to handle geometric data of size and complexity that is essentially hopeless for grid-based techniques.

Acknowledgements

Doing a PhD has been similar in many ways to taking a random walk through a maze: it took the better part of two years to stumble upon a topic that was intriguing yet underexplored. From there, it has taken another four years to unravel some of its mysteries, with a lot more yet to be uncovered! I am eternally grateful to my mentors, collaborators, family and friends who have taken this long but rewarding journey with me.

First and foremost, I would like to thank my PhD advisor, Keenan Crane, for his boldness which inspired me, his unrelenting belief which helped me stay the course, and his patience which supported me when I needed it most. Keenan has taught me so much about being driven by curiosity, not being afraid to take "big bets," and communicating ideas effectively. These learnings will stay with me for life. I would also like to express my deepest gratitude to Bailey Miller, Ioannis Gkioulekas, Dario Seyb and Wojciech Jarosz for being absolutely amazing collaborators and friends. The enthusiasm you have shared, the time you have invested, and the hardwork you have put into this research has opened new doors that I, at times, did not think existed. I feel extremely lucky to have been a part of teams that have functioned so well together, and from whom I have learned so much. I truly enjoyed our collaborations, and I look forward to many more in the future!

I would like to thank Matt Pharr, Rasmus Tamstorf, Michael Mascagni, Gautam Iyer, Matthew O'Toole and Bradley Rothenberg for their wholehearted support over the course of this journey. Your insights and encouragement have gone above and beyond anything I could have asked for.

The Geometry Collective, and Carnegie Mellon more generally, provided a fantastic environment to work and build friendships in. It was a joy to have spent time with Nick Sharp, Chris Yu, Kai Ye, Mark Gillespie, Etienne Corman, Nicole Feng, Olga Guțan, Hossein Baktash, Yousuf Soliman, Mina Konakovic, Nimo Wode Ni, Josua Sassen, Max Slater, Daniel Li, Ethan Lu, Vidya Narayanan, Ravi Mullapudi, Evan Shimizu, Karima Ma, Arjun Teh, Jim McCann, Kayvon Fatahalian, Angela Jiang, John Wright, Nika Haghtalab, Naama Ben-David, David Wajc, Colin White, Noam Brown, Nic Resch, Tanya Marwah, and Gaurav Pathak. I would like to give a special shoutout to Christina Contreras and Deb Cavlovich, who have personally ensured over the years that everything runs smoothly for all students at CMU, myself included.

During my PhD, I was fortunate to have interned with three top-notch teams who helped greatly broaden my perspective: thank you Matt Pharr, Chris Wyman, Daqi Lin, Markus Kettunen, Benedikt Bitterli, Ravi Ramamoorthi, Marco Salvi, Eugene d'Eon, Lifan Wu, Aaron Lefhon, and Ken Museth for welcoming me at NVIDIA, and teaching me so much more about rendering and physics. Thank you Noam Aigerman, Danny Kaufman, Vova Kim, and Nathan Carr for deepening my understanding of geometry processing at Adobe, and thank you Bradley Rothenberg, Blake Courter, Suraj Musuvathy, Johann Korndoerfer, and Trevor Laughlin for giving me the opportunity to experience the exciting and fast-paced world of startups at nTop!

Prior to starting my PhD, I have my undergraduate and high school advisors, Eitan Grinspun, Michael Reed and Vipul Mehra, to thank for captivating me with their infectious excitement and enthusiasm for physics and computer graphics. I would also like to thank my friends,

Kunal Mehta, Zan Gilani, Hong-Yi TuYe, Bitania Wondimu, Mihika Barua, Vishal Alluri, Alex Xu, Kareem Carryl, Henrique Maia, Udit Toshniwal, Ishaan Jalan, Veer Bhartiya, Bhavik Merchant, Ankit Vaish, Karan Kaushik, and Akshay Shah, for enduring and encouraging these passions of mine, as well as my colleagues at IrisVR, Shane Scranton, Nate Beatty, Jack Donovon, Robin Kim, George Valdes, Ailyn Mendoza, Ana Garcia, Tyler Holf, Ezra Miller and Matt Gooding, for giving me the opportunity to gain my first real-world work experience.

Last, and most importantly, the *biggest* thank you to my wife and lifelong friend Ellen Vitercik, my loving parents Chhaya and Rajesh Sawhney, my steadfast brother Arjun Sawhney, and my adorable cat Laddu for always being by my side! Through the ups and the downs, you have provided me with a platform like no other to pursue my goals. Words can hardly do justice to the gratitude I feel for having such a supportive family.

A Note About Content

The text and figures in this thesis have significant overlap with the papers published by my collaborators and I over the course of my PhD. The following publications, used with permission, are the original source of much of the content here:

1. Rohan Sawhney and Keenan Crane. Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-Based Methods on Volumetric Domains. *ACM Transactions on Graphics* (2020).

2. Rohan Sawhney , Dario Seyb , Wojciech Jarosz[†], and Keenan Crane[†]. Grid-Free Monte Carlo for PDEs with Spatially Varying Coefficients. *ACM Transactions on Graphics* (2022).

3. Rohan Sawhney , Bailey Miller , Ioannis Gkioulekas[†], and Keenan Crane[†]. Walk on Stars: A Grid-Free Monte Carlo Method for PDEs with Neumann Boundary Conditions. *ACM Transactions on Graphics* (2023).

4. Bailey Miller , Rohan Sawhney , Keenan Crane[†], and Ioannis Gkioulekas[†]. Boundary Value Caching for Walk on Spheres. *ACM Transactions on Graphics* (2023).

5. Bailey Miller , Rohan Sawhney , Keenan Crane[†], and Ioannis Gkioulekas[†]. Walkin' Robin: Walk on Stars with Robin Boundary Conditions. *ACM Transactions on Graphics* (2024).

The symbols and [†] denote equal contribution.

Contents

Contents **9**

1 Introduction **11**
1.1 Motivation . . . 11
1.2 Approach & Scope . . . 13
1.3 Related Work . . . 15
1.4 Contribution . . . 16

2 Background: Differential & Integral Equations **18**
2.1 Partial Differential Equations . . . 18
2.2 Boundary Integral Equation . . . 22
2.3 Feynman–Kac Formula . . . 25

3 Basic PDE Estimators **33**
3.1 Background: Monte Carlo Integration . . . 33
3.2 Walk On Spheres . . . 38

4 Estimators For PDEs With Mixed Boundary Conditions **45**
4.1 Star-Shaped Regions . . . 46
4.2 Walk On Stars With Neumann Conditions . . . 46
4.3 Walk On Stars With Robin Conditions . . . 51

5 Estimators For PDEs With Spatially Varying Coefficients **56**
5.1 Transformations . . . 56
5.2 Algorithms . . . 59
5.3 Extensions . . . 61

6 Solver Implementation & Tuning **63**
6.1 Closest Point Queries For Walk On Spheres . . . 63
6.2 Accelerated Geometric Queries For Walk On Stars . . . 65
6.3 Epsilon Parameter & Convergence Of Estimators . . . 68

7 Variance Reduction **71**
7.1 Importance Sampling Of Source Terms . . . 71
7.2 Control Variates . . . 72
7.3 Tikhonov Regularization . . . 74
7.4 Adaptive Sampling And Denoising . . . 74
7.5 Boundary Value Caching . . . 75
7.6 Reverse Random Walks . . . 82

7.7 Weight Window 84

8 Evaluation & Comparisons 86
8.1 Geometric Robustness, Scalability & Flexibility 86
8.2 Comparison With Traditional Grid-Based PDE Solvers 92
8.3 Comparison With Alternative Grid-Free Monte Carlo Solvers 97

9 Future Directions 101
9.1 Enhancing Solver Ef ciency 101
9.2 Broadening Solver Applicability 103

Bibliography 104

A Green's Functions & Their Derivatives 124
A.1 Poisson Equation 124
A.2 Screened Poisson Equation 126

B Open Domains & Double-Sided Boundaries 130
B.1 Walk On Stars 130
B.2 Boundary Value Caching 131

C Operator-Theoretic Analysis 132
C.1 Background 132
C.2 Walk On Stars With Dirichlet-Neumann Conditions 133
C.3 Walk On Stars With Robin Conditions 134

D Re ectance & Radius Bound In 2D Domains 135

E Girsanov Transformation 136
E.1 Application To PDEs 136
E.2 Chain Rule Of Stochastic Calculus 136
E.3 Derivation Of Eq. 5.3 137

F Pseudocode: Accelerated Geometric Queries For Walk On Stars 138

Chapter 1

Introduction

Figure 1.1: Grid-free Monte Carlo methods developed in this work can signi cantly speedup the engineering design cycle by eliminating the major bottleneck of discretization in conventional solvers for partial differential equations (left). Lifting the dependence on discretization and global solves improves robustness, and like Monte Carlo ray tracing allows computation effort to be focused entirely on local regions of interest (right).

1.1 Motivation

The ability to accurately model and analyze large amounts of geometric information is fundamental to many scienti c and engineering disciplines, ranging from geology to medicine, and autonomous driving to industrial design. Subtle differences in ne-scale geometry can have a major impact on the large-scale behavior of many physical systems—consider, for instance, the in uence of millions of tiny alveolar air sacs on the dispersion of oxygen in the lungs, the impact of intricate grille patterns on the acoustic performance of a microphone, and the role of all the wiring and plumbing in a building information model on the thermal response of a structure. At all scales, detailed and irregular geometry plays a crucial role in our ability to understand the function of a physical system, evaluate its performance and predict its failure modes.

Techniques based on partial differential equations (PDEs) provide powerful tools for analyzing many such physical phenomena. Unfortunately, despite the drastic increase in our ability to capture and generate complex geometric models in recent years, conventional methods for solving PDEs are not yet at a stage where they "just work" on problems of real-world complexity. Basic tasks involving PDEs still entail careful preprocessing or parameter tuning, and solvers regularly exhibit poor scaling in time or memory. Even more broadly, models of real physical systems must often integrate disparate phenomena, such as light transport and heat transfer, which classically demand very different computational tools that do not "play well together". For these reasons, there remains a large divide between our ability to *visualize* and *simulate* the natural world, and we tend to shy away from simulating it at its original level of complexity by either making gross approximations via model reduction and homogenization—or by tempering our ambition.

Figure 1.2: Real-world geometry has rich surface detail (left) and intricate internal structure (center). On such domains, FEM-based geometric algorithms struggle to mesh, setup, and solve PDEs—in this case taking more than 14 hours and 30 GB of memory just for a basic Poisson equation. Our Monte Carlo solver uses less than 1 GB of memory and takes less than a minute to provide a preview (center right) that can then be progressively refined (far right).

A significant issue with traditional numerical methods for solving PDEs, such as the finite element method (FEM), is the end-to-end cost of the pipeline: even if the FEM solver is fast, one must first convert the boundary description of the input geometry into a "simulation-ready" volumetric mesh. Unfortunately, meshing is brittle and often requires intervention from expert engineers: it can easily fail on data with minor imperfections such as self-intersections, and a few badly-shaped elements can spoil an entire FEM solution. Furthermore, state-of-the-art robust meshing algorithms can be extremely time consuming and memory intensive on intricate geometric domains (e.g., Fig. 1.2, 8.1 & 8.4). Though performance can be improved with more powerful processors, the inherently sequential nature of mesh generation makes it difficult to truly leverage increasingly parallel architectures. Moreover, even when meshing succeeds, important geometric detail can be lost in the discretization (Fig. 1.3 & 8.1), which results in aliasing artifacts in the PDE solution (Fig. 8.9 & 8.15).

Figure 1.3: The bottleneck in conventional PDE solvers like FEM is often not the solve itself, but rather the cost of meshing (top). Robust meshing algorithms [105] can also sacrifice spatial detail–here destroying key features like blood vessels (bottom).

The cost and difficulty of mesh generation for conventional FEM sparked the development of meshless FEM and boundary element methods (BEM), though all of these approaches suffer from a common problem: the need to spatially discretize (e.g., mesh or point-sample) the domain interior. BEM must be integrated with volumetric methods like FEM to han-

Figure 1.4: Unlike FEM and Monte Carlo, traditional BEM does not consider volumetric functions (i.e., coefficients and source terms) that affect the PDE solution.

Figure 1.5: So-called “meshless” methods still perform a process akin to global meshing, which can result in spatial aliasing of fine features. One ends up with a mesh-like structure that must satisfy stringent sampling criteria to avoid numerical blowup, and must solve a large globally coupled linear system. Figure adapted from [109, Figure 6].

dle interior terms (Fig. 1.4), while even so-called meshless methods must carefully place nodes over the entire domain (Fig. 1.5). Furthermore, for problems with volumetric heterogeneities, the discretization must be carefully adapted to regions where material coefficients exhibit fine detail (Fig. 5.1 & 8.10). As a result, many engineering and scientific disciplines today suffer from non-interactive workflows due to the bottleneck of needing to first convert large amounts of geometric information into a form suitable for PDE-based analysis and simulation (Fig. 1.1, left). This limits experts and non-experts alike from making sense of their data, and analyzing it quickly to solve geometric problems.

1.2 Approach & Scope

To address these challenges, this thesis makes a major break from conventional PDE solvers, and instead explores how to solve basic, yet fundamental PDEs in geometric computing with grid-free Monte Carlo methods based on Muller's walk on spheres (WoS) algorithm [174]. This shift mirrors an analogous development in the 1990s for photorealistic rendering: for reasons nicely summarized by Jensen et al. [116], algorithms built around finite element radiosity [85] gave way to Monte Carlo ray tracing of the light transport equation [121]. A key motivation was to simulate more complex illumination, but the shift also made it possible to work with scenes of extreme geometric complexity—modern renderers today can handle trillions of effective polygons [76] and, in stark contrast to FEM, provide high-quality results without any preconditions on the input geometry. As all geometric computation boils down to simple ray intersection queries, Monte Carlo renderers also work with representations other than polygon meshes. They exhibit excellent scaling, offer a trivial parallel implementation, and allow for view-dependent evaluation. Collectively, these features of Monte Carlo rendering have helped it revolutionize industries such as film, architecture and industrial design by enabling practitioners to iterate on their designs quickly.

Figure 1.6: The basic idea behind walk on spheres is that at any point x, the value of a harmonic function $u(x)$ equals the average over a sphere around x [9]. Hence we recursively take a single Monte Carlo sample to estimate this average until we hit the boundary. No spatial discretization is needed since the largest empty sphere can be determined using a closest point query.

Figure 1.7: As with Monte Carlo ray tracing, grid-free Monte Carlo methods like walk on spheres can solve PDEs directly on a variety of geometric representations without needing to discretize the boundary or volume. Unlike traditional PDE solvers, WoS also exactly captures discontinuous boundary conditions and provides a meaningful solution even on geometry with poor element quality; the solution degrades gracefully in the presence of noise.

The Monte Carlo methods described in this work provide an analogous approach for geometric computing: the main idea is to express elliptic PDEs, arising in countless physical problems such as the diffusion of heat, electrostatic potentials, and incompressible uid ows, as recursive integral equations that look a lot like the rendering equation [121]. This reformulation enables use of Monte Carlo integration to solve these integral equations. Samples are generated by replacing recursive ray tracing with the recursive walk on spheres algorithm (Fig. 1.6) and its generalizations, which use the uniform distribution over a sphere to exactly model large steps of continuous random processes such as Brownian motion [191]. This approach shares a number of bene ts with Monte Carlo rendering:

- Geometric Flexibility. It can work directly with implicit surfaces, NURBS/subdivision surfaces, constructive solid geometry, procedural/instanced geometry, etc., without explicit tessellation (Fig. 1.7).
- Geometric Robustness. Geometry need not be watertight, manifold, nor free of self-intersections; sharp edges, small details, and thin features are exactly preserved.
- Scalability. The main cost is a bounding volume hierarchy (BVH) for distance queries, which is $O(n \log n)$ in time and memory with respect to the size of the boundary (Fig. 1.8).
- Parallelism. It is trivial to achieve near-perfect parallel scaling, and many operations are easily vectorized.
- Correctness. Since there is no discretization of space or approximation of function spaces, one obtains the exact solution in expectation, i.e., error is almost entirely due to the variance of the Monte Carlo estimator, and can be reduced by simply taking more samples.
- Adaptivity. Adaptive sampling akin to radiance caching [266] can signi cantly reduce cost in smooth regions (Fig. 7.7); progressive sampling enables rapid previews of PDE solutions (Fig. 1.2 & 4.2).
- Output Sensitivity. The solution can be evaluated in local regions of interest, like a small window (Fig. 1.9 & 6.1) or a slice (Fig. 8.6), without having to rst perform a global solve.
- Compatibility. Monte Carlo methods t easily into standard pipelines for geometric computing, as they can be used as “black box” solvers that return reliable and accurate solution values at any given query point (e.g, at mesh vertices).

Figure 1.8: Finite element methods exhibit unpredictable performance, as models with simple geometry but poor element quality (left) can confound even robust meshing algorithms (center, via Hu et al. [105]). The Monte Carlo approach only needs to build a standard bounding volume hierarchy (right), which dramatically reduces precomputation time for solving PDEs.

Monte Carlo methods are not, however, a silver bullet. On simple domains with smooth boundary conditions, solvers like FEM are quite mature and hard to beat in terms of solve time; here Monte Carlo can be slow to eliminate high-frequency noise (though see Ch. 7 for variance reduction strategies). Yet for more complex problems, end-to-end performance depends on many factors beyond just the rate of convergence of the core solve, such as mesh generation, parallel scaling, and visualization; Ch. 8 provides an in-depth discussion on tradeoffs with conventional PDE solvers. It may also not be straightforward to formulate a Monte Carlo estimator for any given PDE. Here we do not strive for full feature compatibility with traditional solvers and all the PDEs they can solve—we instead focus on a specific class of problems, namely 2nd order linear elliptic equations (reviewed in Ch. 2), which power a large array of applications in geometric computing such as surface reconstruction [129, 130] and shape optimization [204] (using measurements of physical quantities like temperature [277]). Monte Carlo solvers are not limited just to WoS [101, 134, 172, 217]—we compare against some of these techniques in Ch. 8.

On the whole, we find that Monte Carlo methods have a number of attractive computational features that make them well-suited to PDE-based geometric computing. By avoiding the daunting challenge of mesh generation, these methods offer a framework that is scalable, parallelizable, easy-to-tune, and numerically robust (Fig. 1.1, right). They also occupy a unique place in the broader landscape of numerical solvers for PDEs, as they complement the strengths of existing grid-based techniques (e.g, local versus global evaluation of solutions).

1.3 Related Work

The biggest issue preventing broader adoption of grid-free Monte Carlo methods like WoS (discussed in detail in Sec. 3.2) is that estimators have been developed only for a narrow set of PDEs beyond the original Laplace Dirichlet problem studied by Muller [174]—these include the Poisson and screened Poisson equations [51, 61], the heat equation [48, 95], the biharmonic equation [84, 154] and certain (mildly) nonlinear PDEs [28]. WoS has previously been used for specific problems in, e.g, molecular dynamics [160], integrated circuit design [146], porous media [109], and electrostatics [108]. A few variants of WoS have also been developed, such as walking on rectangles [49, 216], on the boundary [217, 242], off-centered walks [110], and Green's function first passage (GFFP) methods [81, 107]; we draw on some of these techniques in this work.

Unfortunately, most prior work on grid-free Monte Carlo methods lacks thorough numerical evaluation, and even then only considers very simple geometry such as a box or cylinder. Moreover, questions essential for real applications like how to solve larger classes of equations, deal

with defective or highly detailed geometry, reduce variance and build high-performance systems have not been sufficiently explored. Despite some very close analogs in photorealistic rendering [147, 171], researchers and practitioners in high-performance computing also seem to be largely unaware of grid-free methods for PDEs. It is for this reason that we find this approach exciting: the computer graphics community has made Monte Carlo integration the workhorse of photorealistic rendering, and we believe that it can have a similar impact on PDE-based techniques which are central to computer-aided design and engineering.

Since our initial publication which introduced WoS to computer graphics [221], we have designed robust and efficient WoS estimators for a much broader set of fundamental PDEs [167, 223, 223, 224], which we will detail over the course of this text. Other graphics researchers have followed suit, and concurrently expanded the capabilities of WoS to, e.g, support exterior and time-dependent problems [47, 179], simulate fluids and the coupled physics of radiation and diffusion [14, 211], solve inverse problems via differentiable formulations [272], and improve estimation quality through variance reduction [11, 151, 206].

1.4 Contribution

We develop a holistic Monte Carlo framework encompassing integration, variance reduction and accelerated geometric queries to solve linear elliptic PDEs in volumetric domains, i.e., an N-dimensional solid region in $\mathbb{R}^N$ (we focus on $N = 2$ and 3). In particular, we provide:

1. a unified discussion of previously developed WoS estimators for elliptic PDEs (Ch. 3).
2. a new grid-free method called walk on stars (WoSt) which generalizes WoS to solve PDEs with any arbitrary mix of Dirichlet, Neumann and Robin boundary conditions (Ch. 4)—these boundary conditions are a basic component of virtually every real physical system.
3. a new integral formulation and subsequent WoS estimators for PDEs with variable material coefficients, by establishing a close connection with null-scattering techniques [188] for rendering heterogeneous participating media (Ch. 5).
4. easy-to-implement estimators for spatial derivatives of PDE solutions (Ch. 3 & 5).
5. Efficient BVH-based geometric queries to accelerate the above estimators (Sec.6.2).
6. several variance reduction strategies (Ch. 7), including:
   a) importance sampling of source terms, adaptive sampling of the solution, and control variates for spatial derivatives.
   b) a boundary value caching (BVC) scheme, similar in spirit to virtual point light methods in rendering [46, 132], that greatly amortizes the cost of long walks and suppresses the typical salt-and-pepper noise characteristic of independent Monte Carlo estimates.
   c) a reverse WoSt estimator that splats known boundary and source data to multiple points in the interior of a domain, by extending the recently developed bidirectional formulation of WoS [206] from PDEs with Dirichlet conditions to those with Neumann and Robin conditions as well.
   d) a weight window strategy from neutron transport [25, 102] that significantly reduces noise and improves efficiency in problems with high-frequency material coefficients.

Figure 1.9: Thermal analysis of NASA's Curiosity Mars rover. Keeping temperatures within speci ed thermal limits is critical to mission success—but thermal modeling is historically dif cult to integrate into the design phase, due to intricate geometry not easily captured via nite element models (Fig. 1). Here our walk on stars solver (Ch. 4) computes realistic temperature estimates quickly and progressively even for extremely complex geometry, without needing to volumetrically mesh the domain. A “deferred shading” approach provides output-sensitive evaluation, computing temperature values only at the points visible in screen space (top right). We can hence analyze temperature in local regions of interest, without computing a global solution (bottom row).

Our approach connects to a large body of work on Monte Carlo rendering [203]. From the PDE point of view, the major difference is that the differential equation governing radiative transfer is rst order in space, whereas the PDEs we seek to solve have second order, diffusive terms that demand different numerical techniques. There are of course many parallels between these problems from a mathematical, computational, and system design point of view, which we explore throughout this text. More broadly, by framing PDE-based geometric computing in terms of stochastic processes, we build a bridge to rich tools not just from Monte Carlo rendering, but also reinforcement learning, stochastic control and mathematical nance. We evaluate the capabilities of our solvers in analyzing physical systems with complex geometry (see e.g, Fig. 1.9) in Ch. 8, and end with a discussion on future directions in Ch. 9.

Chapter 2

Background: Differential & Integral Equations

We rely on concepts from the theory of partial differential equations, integral equations and stochastic calculus to develop Monte Carlo algorithms for solving PDEs. We provide essential background here as few readers will be familiar with all of these topics, and scope out the speci c types of PDEs we will consider in this work. Chapter 3 then describes the Monte Carlo walk on spheres algorithm, which is the starting point for the generalizations of WoS we develop in Ch. 4 - 7 for solving these equations.

2.1 Partial Differential Equations

Many natural phenomena are described by relating rates of change. An ordinary differential equation (ODE) describes how a quantity changes in time. For instance, $d^2p(t)/dt^2 = g$ implicitly describes the trajectory of a particle $p(t)$ under the in uence of a gravitational acceleration g. Only by solving this ODE for $p(t)$ do we obtain an explicit description of the trajectory. Likewise, a PDE implicitly describes a function via relationships between partial derivatives in space. A prototypical example is the Laplace equation $d^2u(x,y,z)/dx^2 + d^2u(x,y,z)/dy^2 + d^2u(x,y,z)/dz^2 = 0$ (or more compactly $Du = 0$), which describes the steady-state of a diffusion process, i.e., the way heat diffuses smoothly from the domain boundary into the interior. Just like ODEs, we must ultimately solve for an explicit function u satisfying this relationship. Since most PDEs do not have analytical solutions (including the Laplace equation), we rely on numerical methods to solve them.

Given the vast number of PDEs and the diverse phenomena they model such as thermodynamics, electromagnetism, uid mechanics, elasticity, quantum mechanics, and even the movement of stock prices, numerical methods are often specialized to consider speci c classes of equations. PDEs can be classi ed based on their form and properties, as we describe below:

Order And Linearity. The order of a PDE refers to the highest-order degree of any derivative appearing in the PDE. For instance, the Laplace equation is 2nd order as it involves spatial derivatives no higher than degree two, whereas a biharmonic equation $D^2u = 0$ is 4th order. A PDE is linear if it is a linear polynomial in the function and its derivatives. The Laplace equation is linear, but the inviscid Burger's equation $\frac{\partial}{\partial t}u(x,t) = u(x,t)\frac{\partial}{\partial x}u(x,t)$ is nonlinear as it multiplies the function by one of its derivatives. We will consider 2nd order linear PDEs in this text.

Ellipticity. Roughly speaking, elliptic equations are those whose solutions are captured by the idea of “repeated local averaging”. These PDEs often describe steady-state processes, like the distribution of temperature in a stationary object. The Laplace equation is elliptic, as its solution at any point in a domain W equals the average value in some small neighborhood (i.e., the mean value property see Eq. 2.13). Parabolic PDEs instead describe time-dependent processes, like the changing temperature in an object over time. The canonical example is the heat equation $Du(x,t) = \partial u(x,t)/\partial t$, whose solution u becomes progressively smoother and eventually reaches an

Figure 2.1: A non-negative Robin coefficient m linearly interpolates between Dirichlet ($m = \infty$) and Neumann ($m = 0$) boundary conditions, which prescribe solution values and derivatives on the boundary, respectively.

equilibrium described by the Laplace equation as $t \to \infty$. Hyperbolic PDEs, on the other hand, are often associated with processes that involve wave propagation, such as sound waves.

We will focus on elliptic problems in this work, as their solutions can be computed by taking averages of local estimates via Monte Carlo integration (Sec. 3.1). Formally, a 2nd order linear PDE with constant coefficients is elliptic if its *principal symbol* (i.e., the polynomial corresponding to its highest-order term) is greater than zero away from the origin. For PDEs with variable coefficients such as $\nabla \cdot (K(x) \nabla u) = 0$, ellipticity requires the diffusion matrix $K(x) : \Omega \mapsto \mathbb{R}^{N \times N}$ to be positive definite, i.e., $\sum_{i,j=1}^{n} K^{ij}(x) h_i h_j > 0$ for all non-zero $h \in \mathbb{R}^N$ [64, 72].

Boundary Conditions. PDEs are often paired with an additional set of constraints called *boundary conditions*. Boundary conditions can be used to specify various physical constraints such as temperatures, voltages, forces and velocities on the boundary $\partial\Omega$ of a domain $\Omega \subset \mathbb{R}^N$, and typically have a major impact on the solution of a PDE. They come in many flavors: for instance with a Laplace equation, Dirichlet boundary conditions specify the solution value along $\partial\Omega$ (e.g, a surface held at a fixed temperature), which uniquely determines the solution inside Ω. Neumann conditions instead specify the value of the normal derivative of the solution along $\partial\Omega$ (e.g, the heat flux across a surface); the solution inside Ω is determined only up to an additive constant. Robin conditions linearly combine both solution values and their derivatives on $\partial\Omega$, which means that the PDE solution interpolates between the solution with pure Dirichlet conditions and pure Neumann conditions inside Ω (Fig. 2.1), and is uniquely determined. More generally, one can prescribe different boundary conditions on disjoint parts of $\partial\Omega$. For a Laplace equation, this yields a boundary value problem (BVP) of the form

$$\begin{aligned} \Delta u(x) &= 0 && \text{on } \Omega, \\ u(x) &= g(x) && \text{on } \partial\Omega_D, \\ \frac{\partial u(x)}{\partial n_x} &= h(x) && \text{on } \partial\Omega_N, \\ \frac{\partial u(x)}{\partial n_x} + m(x) u(x) &= \ell(x) && \text{on } \partial\Omega_R, \end{aligned} \tag{2.1}$$

where the boundary is partitioned into a Dirichlet part $\partial\Omega_D$ with prescribed values $g : \partial\Omega_D \to \mathbb{R}$, a Neumann part $\partial\Omega_N$ with prescribed derivatives $h : \partial\Omega_N \to \mathbb{R}$, and a Robin part $\partial\Omega_R$ with prescribed right hand side $\ell : \partial\Omega_R \to \mathbb{R}$. Here n_x is the unit outward normal to $\partial\Omega$ at x, and $m \in \mathbb{R}_0$ is a non-negative Robin coefficient that can vary over $\partial\Omega_R$ (negative values of m occur less frequently in natural physical systems [86, Sec. 1], and we do not consider them here). Since Robin conditions serve as general first-order boundary conditions, we recover Neumann conditions when $m = 0$, and Dirichlet conditions as $m \to \infty$. A function u is harmonic if it satisfies Eq. 2.1 on Ω. We will provide Monte Carlo estimators for BVPs with any combination of Dirichlet, Neumann and Robin boundary conditions in Ch. 3 & 4.

Exterior Problems, Open Domains And Double-Sided Boundary Conditions. One may also solve PDEs in the exterior of a domain, i.e., $\mathbb{R}^N \setminus \Omega$. The domain Ω need not be closed/watertight, and can have different boundary conditions prescribed on either side of $\partial\Omega$. We will describe how to handle such generalizations with our method in Sec. 3.2.4 and App. B.

2.1.1 Linear Elliptic Equations

Beyond a basic Laplace equation which essentially interpolates boundary values, 2nd order linear elliptic PDEs can also model rich spatially varying material properties of a medium. In thermodynamics, for example, PDEs with variable coefficients model how heterogeneous composite materials conduct or insulate heat—much as early algorithms for photorealistic rendering were motivated by predictive lighting design [144], such models can be used to predict and improve thermal efficiency in building design [276]. Likewise, variable permittivity in electrostatics impacts the design of antennas [194] and the simulation of biomolecules [66]; in hydrology, variable transmissivity of water through soil impacts remediation strategies for groundwater pollution [269]. More directly connected to our work, variable coefficients in the light transport equation are used to model heterogeneity in participating media [188]. Beyond spatially varying materials, variable coefficients can also be used to model curved geometry by using PDE coefficients on a flat domain to encode an alternative Riemannian metric (Fig. 8.8). Below, we discuss generalizations of the Laplace equation we will consider in this text:

Source term. Continuing with the heat analogy for a Laplace equation, a source term $f : \Omega \to \mathbb{R}$ adds additional "background temperature" to a PDE (Fig. 2.2, center left). For instance, a Poisson equation has the form

$$\Delta u(x) = f(x) \quad \text{on } \Omega \tag{2.2}$$

subject to Dirichlet, Neumann or Robin boundary conditions.

Diffusion. The rate of diffusion in a spatially varying medium is modeled by replacing the operator Δ with $\nabla \cdot (k(x)\nabla \cdot)$, where $k : \Omega \to \mathbb{R}_{>0}$ is the diffusion coefficient (Fig. 2.2, center).

Drift. A drift coefficient, given by a vector field $\vec{w} : \Omega \to \mathbb{R}^N$, models the motion of a material in a particular direction. For instance, the steady-state advection equation $\vec{w}(x) \cdot \nabla u(x) = 0$ describes a quantity u that is unchanged as it flows along $\vec{w}$; adding this term to a Poisson equation causes heat to drift as it diffuses (Fig. 2.2, center right).

Absorption. An absorption (or screening) coefficient $s : \Omega \to \mathbb{R}_{>0}$ models "cooling" of the solution due to the background medium; larger coefficient values dampen the solution more. E.g., a screened Poisson equation (seen in Fig. 2.2, far right) is given by

$$\Delta u(x) - s(x)u(x) = f(x) \quad \text{on } \Omega, \tag{2.3}$$

again subject to boundary conditions.

Combining all these terms and coefficients then yields a linear elliptic equation of the form

$$\nabla \cdot (k(x)\nabla u(x)) + \vec{w}(x) \cdot \nabla u(x) - s(x)u(x) = f(x) \quad \text{on } \Omega. \tag{2.4}$$

Here we do not require the source term $f : \Omega \to \mathbb{R}$ to be continuous, but we will assume the diffusion coefficient $k : \Omega \to \mathbb{R}_{>0}$ is twice-differentiable, the drift coefficient $\vec{w} : \Omega \to \mathbb{R}^N$ is a

Figure 2.2: Effect of each spatially-varying term of Eq. 2.4 on the PDE solution.

vector field expressible as the gradient of a scalar field, and the absorption coefficient $s : W \to \mathbb{R}_0$ is continuous (C^0). Though these coefficients can be generalized further (for instance, k can be any arbitrary positive definite matrix and s can be negative), the conditions we impose on them are sufficient to ensure ellipticity [64, 72]. Moreover, these conditions will enable us to derive Monte Carlo estimators for variable-coefficient equations in Ch. 5 that do not have to resort to numerical homogenization [52], and can directly resolve the original, detailed solution (see, e.g, Fig. 5.1).

2.1.2 Fundamental Solutions

A Green's function captures the influence of the source term f on the solution of a linear elliptic equation with constant coefficients. In particular, it describes the (fundamental) solution when the source is a Dirac delta distribution δ_x centered at a single point $x \in W$. For instance, in the case of Eq. 2.2, the Green's function $G^W(x, y)$ is the solution to the Poisson equation $\Delta u(y) = -\delta_x(y)$. In general, Green's functions will depend on the shape of the domain W and the choice of boundary conditions—as a result, they are typically not known in closed-form. However, explicit expressions are available for important special cases, e.g, the free space Green's function $G^{\mathbb{R}^N}$ on $W = \mathbb{R}^N$, and the Green's function G^B for a ball $W = B$ with zero-Dirichlet boundary conditions (App. A). The walk on spheres and walk on stars algorithms we describe in the following chapters will effectively provide a bridge between closed-form Green's functions on special domains, and solutions to PDEs on more general domains.

Figure 2.3: The Green's function and its normal derivative are singular at the point they are centered on, but decay smoothly and fall-off quickly away from the singularity.

The Poisson kernel likewise captures the influence of the boundary conditions on the solution, e.g, when the Dirichlet function g is a Dirac delta distribution δ_x centered on a single boundary point $x \in \partial W$. At any point $y \in W$ with associated normal n_y, it can be expressed as the normal derivative of a Green's function:

$$P^W(x, y) := \frac{\partial G^W(x, y)}{\partial n_y}. \tag{2.5}$$

As with Green's functions, common Poisson kernels are known explicitly in $\mathbb{R}^N$ and for a ball.

2.2 Boundary Integral Equation

One can often reformulate linear elliptic PDEs as recursive integral equations, akin to the classic rendering equation [121]. As in rendering, these equations can be solved without discretizing space, by recursively applying Monte Carlo integration (as will be discussed in Ch. 3). In this section, we provide a boundary integral formulation for PDEs with constant coefficients (e.g, Eq. 2.2). Section 2.3 then describes a stochastic integral formulation for Eq. 2.4 with varying coefficients.

Derivation. The solution u to a Poisson equation $\Delta u = f$ can be expressed via an integral involving the associated Green's function and Poisson kernel. Assume for now that the domain Ω is watertight with smooth boundary $\partial\Omega$, and let x be an evaluation point in the interior of Ω. We first multiply the Poisson equation with its Green's function G^Ω, and integrate over Ω to get

$$0 = \int_\Omega G^\Omega(x,y)\, \Delta u(y)\, dy + \int_\Omega G^\Omega(x,y)\, f(y)\, dy. \tag{2.6}$$

Applying integration by parts to the first integral, we have

$$0 = \int_{\partial\Omega} G^\Omega(x,z)\, \frac{\partial u(z)}{\partial n_z}\, dz - \int_\Omega \nabla G^\Omega(x,y) \cdot \nabla u(y)\, dy + \int_\Omega G^\Omega(x,y)\, f(y)\, dy. \tag{2.7}$$

Applying integration by parts again to the second integral, and rearranging terms then yields

$$\int_\Omega u(y)\, \Delta G^\Omega(x,y)\, dy = \int_{\partial\Omega} \frac{\partial G^\Omega(x,z)}{\partial n_z}\, u(z) - G^\Omega(x,z)\, \frac{\partial u(z)}{\partial n_z}\, dz - \int_\Omega G^\Omega(x,y)\, f(y)\, dy. \tag{2.8}$$

From the definitions $\Delta G^\Omega(x,y) = -\delta_x^\Omega(y)$ and $P^\Omega(x,z) = \partial G^\Omega(x,z)/\partial n_z$, we arrive at

$$u(x) = \int_{\partial\Omega} P^\Omega(x,z)\, u(z) + G^\Omega(x,z)\, \frac{\partial u(z)}{\partial n_z}\, dz + \int_\Omega G^\Omega(x,y)\, f(y)\, dy. \tag{2.9}$$

This equation determines the solution u at x entirely through the solution values $u(z)$ and normal derivatives $\partial u(z)/\partial n_z$ on the boundary $\partial\Omega$, and the source values $f(y)$ inside the domain Ω. From Eq. 2.1, the Dirichlet, Neumann and Robin parts of the boundary have prescribed values g, h and k, respectively, while f is specified inside the domain for a Poisson equation. To use Eq. 2.9, we must then determine unknown solution values $u(z)$ on the Neumann boundary $\partial\Omega_N$, unknown derivative values $\partial u(z)/\partial n_z$ on the Dirichlet boundary $\partial\Omega_D$, and either $u(z)$ and $\partial u(z)/\partial n_z$ on the Robin boundary $\partial\Omega_R$, via the relation $\partial u/\partial n + m u = \ell$ when $m > 0$.

2.2.1 General Setting

In practice, Eq. 2.9 cannot be used directly since the Green's function and Poisson kernel for an arbitrary domain Ω are unknown. Fortunately, this equation can be generalized to the boundary integral equation (BIE) [41, Section 2] where these functions are no longer tied to the domain Ω. Instead one may use, e.g, the closed-form Green's function and Poisson kernel for a ball or for $\mathbb{R}^N$. Moreover, while we ultimately seek a solution on Ω, the BIE applies to arbitrary subdomains in Ω:

Boundary Integral Equation

For any two sets $A \subset W$ and $C \subset \mathbb{R}^N$, and for any point $x \in \mathbb{R}^N$, the solution to a Poisson equation satisfies

$$a(x)\, u(x) = \int_{\partial A} P^C(x,z)\, u(z) + G^C(x,z)\, \frac{\partial u(z)}{\partial n_z}\, dz + \int_A G^C(x,y)\, f(y)\, dy, \tag{2.10}$$

where

$$a(x) := \begin{cases} 1, & x \in A, \\ 1/2, & x \in \partial A, \\ 0, & x \notin A. \end{cases} \tag{2.11}$$

Hunter and Pullan [106, Chapter 3.3] provide a derivation. We note that if ∂A is a non-smooth curve in the plane, then $a = 1 - \theta/2\pi$ at a corner with interior angle θ. To keep things simple, we will assume ∂A is smooth, letting $a = 1/2$ at all boundary points.

Though we focus on Poisson equations for simplicity, the BIE extends immediately to screened Poisson equations $\Delta u - su = f$ with constant absorption coefficient $s \in \mathbb{R}_{\geq 0}$: the only modification to Eq. 2.10 is to replace the Green's function and Poisson kernel with their screened counterparts (App. A.2). Likewise, for a constant diffusion coefficient $k \in \mathbb{R}_{>0}$, the Green's function simply scales by a factor k. Constant drift along a fixed direction $\vec{w} \in \mathbb{R}^N$ can be captured via the von Mises–Fisher distribution [75, 215]. Boundary integral equations are also readily available for a variety of other PDEs not directly considered in this work, such as the Helmholtz equation [106, Chapter 3], linear elasticity [106, Chapter 4] and the biharmonic equation [111]. Below, we discuss a few special cases of Eq. 2.10 with different choices for the sets A and C, and describe how the BIE can be generalized to support double-sided boundary conditions in closed and open domains.

Boundary Element Formulation

Conventional numerical solvers like the boundary element method integrate Eq. 2.10 over the PDE domain ($A = W$) using free-space kernels ($C = \mathbb{R}^N$). BEM does not directly support source terms f, leading to the integral

$$a(x)\, u(x) = \int_{\partial W} P^{\mathbb{R}^N}(x,z)\, u(z) + G^{\mathbb{R}^N}(x,z)\, \frac{\partial u(z)}{\partial n_z}\, dz. \tag{2.12}$$

To determine the unknown data u and $\partial u / \partial n$ on ∂W, BEM uses a finite basis of functions (associated with mesh nodes on a discretized boundary) to solve a dense linear system—resulting in the tradeoffs discussed in Sec. 8.2.1.

Mean Value Property Of Harmonic Functions

Monte Carlo methods like walk on spheres (Sec. 3.2) instead integrate the BIE over a ball $B(x,R) \subset W$ of radius R centered at x, adopting kernels from the ball ($A = C = B(x,R)$). At points $z \in \partial B$, these kernels then simplify to $G^B(x,z) = 0$ and $P^B(x,z) = 1/|\partial B|$ (i.e., 1 over the surface area of the ball boundary), yielding the mean value property of harmonic functions

Figure 2.4: Left: As s is increased, the Green's function $G^{s,B}(x,y)$ for a screened Poisson equation becomes more localized around the point x, and the magnitude of the Poisson kernel $P^{s,B}(x,z)$ shrinks. These functions revert to the harmonic Green's function and Poisson kernel as $s \to 0$. Right: The functions $G^{s,B}$ and $P^{s,B}$ are not rotationally symmetric when the point x does not coincide with the center of the ball.

when $f = 0$:

$$u(x) = \frac{1}{|\partial B(x,R)|} \int_{\partial B(x,R)} u(z)\, dz \tag{2.13}$$

This setup greatly simplifies the BIE by eliminating dependence on $\partial u / \partial n$. Unlike BEM, a non-zero source term f is accounted for by adding the integral

$$\int_{B(x,R)} G^B(x,y)\, f(y)\, dy. \tag{2.14}$$

More importantly, WoS evaluates 2.13 by recursively estimating $u(z)$ on ∂B. It therefore does not need to discretize the domain W or its boundary ∂W, nor solve a global system of equations.

Off-Centered Mean Value Property

The point of evaluation x in Eq. 2.13 need not coincide with the center $c \in W$ of a ball $B(c,R)$. With a screened Poisson equation, this leads to the following more general off-centered formulation [55, 110] of the mean value property:

$$u(x) = \int_{\partial B(c,R)} P^{s,B}(x,z)\, u(z)\, dz + \int_{B(c,R)} G^{s,B}(x,y)\, f(y)\, dy. \tag{2.15}$$

We provide explicit expressions for off-centered versions of the functions $G^{s,B}$ and $P^{s,B}$ in App. A.2.2. As shown in Fig. 2.4, the Poisson kernel $P^{s,B}(x,z)$ reduces to $1/|\partial B(x,R)|$ when $x = c$ and $s = 0$, recovering the usual mean value property. In Sec. 5.2.2, we will use Eq. 2.15 to design a WoS estimator for solving variable-coefficient PDEs.

Double-Sided Boundary Conditions

The BIE for double-sided boundary conditions in an open domain $W \subset \mathbb{R}^N$ is given by [41]

$$a(x)\, u(x) = \int_{\partial W} P^+(x,z)\, \left[u^+(z) - u^-(z) \right] + G(x,z) \left[\frac{\partial u^+(z)}{\partial n_z^+} - \frac{\partial u^-(z)}{\partial n_z^-} \right] dz$$
$$+ \int_W G(x,y)\, f(y)\, dy, \tag{2.16}$$

where n^+ and n^- denote unit normals on either side of ∂W (respectively), u^+ and u^- represent corresponding solution values on ∂W, and $P^+(x,z) := \partial G(x,z) / \partial n_z^+$. Since all points are either on the boundary or the domain interior, $a = 1/2$ on ∂W and 1 otherwise. In App. B, we discuss how to apply the walk on stars algorithm for BVPs with Dirichlet, Neumann and Robin boundary conditions (Ch. 4) to open domains and double-sided boundaries.

2.3 Feynman–Kac Formula

In general, we do not have boundary integral representations of PDEs with variable coefficients (like Eq. 2.4), due to the unavailability of Green's functions in $\mathbb{R}^N$, on a ball, or elsewhere. However, the solution to an elliptic PDE can also be described in terms of continuous stochastic processes such as *Brownian motion*, via the Feynman–Kac formula from stochastic calculus [191, Ch. 8]. This formula will provide a critical starting point in Ch. 5 for solving variable-coefficient equations, as it is more general than the BIE from the previous section. Moreover, it has close parallels with volume rendering, providing us with key techniques for numerical integration.

Here we provide essential background on stochastic processes (Sec. 2.3.1) and their associated integral representations of PDEs (Sec. 2.3.2), which at present are not widely used in computer graphics (we refer to Øksendal [191] for a more comprehensive introduction). In particular, the central object that we would like to simulate is Brownian motion, as it captures the phenomenon of diffusion described by elliptic equations through the Feynman–Kac formula. Unfortunately, simulating Brownian motion directly is both expensive and introduces statistical error, especially in domains with complex geometry. We will see later that walk on spheres and its generalizations (Ch. 3 - 5) enable a far more efficient simulation with significantly less statistical error.

2.3.1 Stochastic Processes

A stochastic process is a collection of random variables X_t that represents the evolution of a system over time $t \geq 0$. The process is continuous if it can be observed continuously with time. A key characteristic of a stochastic process is that it incorporates some form of randomness or unpredictability, which means that even if the initial state $X_0 = x$ is known, the future evolution of a process cannot be predicted with certainty. Such a process is therefore typically modelled by a probability density function (PDF) that is non-negative everywhere, and integrates to 1 over the domain on which it is defined. We can use a PDF to calculate the probability with which future random states $X_{t>0}$ of the process take on a permissible range of values. In particular, for a real-valued random variable X_t, integrating the PDF p over an arbitrary interval $[a, b]$ gives the probability that X_t lies inside the interval:

$$P\{a \leq X_t \leq b\} = \int_a^b p(x)\, dx. \tag{2.17}$$

A cumulative density function (CDF) $P(x)$ represents the probability that X_t takes on a value less than or equal to x, i.e., $P(x) := P\{X_t \leq x\}$.

For our purposes, we will consider a continuous time-parameterized family of $\mathbb{R}^N$-valued random variables X_t on a domain A with PDF p^A. The expected value—or mean of all possible values—of any L^1-integrable function $f : A \to \mathbb{R}$ is then given by

$$E[f(X_t)] = \int_A f(x)\, p^A(x)\, dx. \tag{2.18}$$

Throughout, we will informally refer to a realization of a stochastic process as a *random walk*.

Brownian Motion

The central example of a continuous stochastic process is Brownian motion, which is more formally known as a *Wiener process*. Intuitively, a Wiener process describes a random walk by repeatedly taking small Gaussian steps, and letting the variance of the Gaussian go to zero (Fig.

Figure 2.5: Brownian motion can be thought of as the limit of taking small, normally distributed steps.

2.5). This motion is hence isotropic, i.e., there is no preferred direction, and it has no "memory", i.e., the direction of motion at each point in time is independent of all past motion. More formally, a (multidimensional) Wiener process starting at a point $x_0 \in \mathbb{R}^N$ is a time-parameterized family of $\mathbb{R}^N$-valued random variables W_t characterized by the following criteria:

- $W_0 = x_0$
- The process has Gaussian increments, i.e., for all $t, s \geq 0$, the increments $W_{t+s} - W_t \sim \mathcal{N}(0, s)$ are normally distributed with mean 0 and variance s.
- These increments are independent of all previous random variables W_r for $0 \leq r \leq t$.
- The process is almost surely continuous with respect to t, i.e., random walks have continuous paths in $\mathbb{R}^N$.

Diffusion Process

Just as Brownian motion will help us model isotropic diffusion, we will use more general stochastic processes to model diffusion that is anisotropic or exhibits drift in a particular direction (Fig. 2.6, center). Such processes can be defined by combining a deterministic velocity with stochastic "noise", modeled via Brownian motion. In particular, the stochastic differential equation (SDE)

$$dX_t = \vec{w}(X_t)\,dt + dW_t \tag{2.19}$$

describes a stochastic process whose increments dX_t behave exactly like Brownian increments dW_t, offset by a deterministic vector $\vec{w}$ (for a more formal treatment of the notation dX_t, see Øksendal [191, Ch. 5]). More generally, a diffusion process is any stochastic process of the form

$$dX_t = \vec{w}(X_t)\,dt + K(X_t)\,dW_t, \tag{2.20}$$

where for each time t and location X_t, $\vec{w}(X_t) \in \mathbb{R}^N$ gives a direction of drift, and $K(X_t)$ is a symmetric positive definite $N \times N$ matrix that controls the rate and directional bias of diffusion

Figure 2.6: Components of a diffusion process. Note that a diffusion process starting at x will not in general have a uniform exit distribution over a sphere, as it is not isotropic like Brownian motion. Left: the diffusion coefficient $k(x)$ modulates the size of random increments. Center: the drift coefficient $\vec{w}(x)$ adds deterministic offsets to the trajectory. Right: the screening coefficient $s(x) > 0$ describes the probability of a random walk being absorbed, whereas $s(x) < 0$ essentially describes emission of new random walks (which we do not consider in this work).

(i.e., the variance and covariance of the random increments). In our setting, we will typically consider a simpler special case, namely

$$dX_t = \vec{w}(X_t)\,dt + \sqrt{k(X_t)}\,dW_t, \tag{2.21}$$

where $k : \mathbb{R}^N \to \mathbb{R}_{>0}$ is a scalar- rather than matrix-valued function that determines the standard deviation of the normal distribution associated with W_t (Fig. 2.6, left). We assume that neither $\vec{w}$ or k depend on time.

Killed Diffusion Process

We can enrich our model of stochastic processes by also allowing a diffusion process to be probabilistically *killed*—intuitively, a random walk realized from such a process can be absorbed into the background medium (Fig. 2.6, right). This process will be essential for making the connection to PDEs with an absorption coefficient $s \in \mathbb{R}_{\geq 0}$. We assume that the probability with which a random walk is absorbed is exponentially decaying for any value of s. Then $p(t) = se^{-st}$ gives the probability density of a killed diffusion process, and for a total time T, the corresponding probability is [203, Sec. 13.3.1]

$$\int_0^T se^{-st}\,dt = 1 - e^{-sT}. \tag{2.22}$$

Here a larger value of s yields a higher probability of absorption.

Restriction To Bounded Domains

So far we've assumed that a diffusion process X_t is free to wander around all of $\mathbb{R}^N$, but suppose we are interested in a particular domain $W \subset \mathbb{R}^N$. A common question we might ask is, "where does X_t first exit the domain W?" In other words, what is the probability that X_t first hits any given point on the boundary ∂W. This information will be central to our approach to solving PDEs, as it corresponds to the Poisson kernel discussed in Sec. 2.1.2.

Figure 2.7: A random walk terminates when it hits an absorbing Dirichlet boundary ∂W_D (left), but is pushed back into the domain along the normal to a re ecting Neumann ∂W_N (center) or Robin boundary ∂W_R (right). The walk continues forever with ∂W_N, but is eventually absorbed on ∂W_R.

In the study of stochastic processes [27], the presence of a boundary ∂W is often described by using another random variable t called stopping time, which is a path-dependent quantity describing when a diffusion process starting at $x \in W$ rst hits the boundary, i.e., $t := \inf\{t \geq 0 : x + X_t \in \partial W\}$. Then, the corresponding exit location X_t on ∂W is a random variable as well, and the probability density of an associated random trajectory of the process in W is given by the Poisson kernel P^W [191, Ch. 7]. One could also consider the distribution of exit times on $\mathbb{R}_{\geq 0}$: though the exit time density is well-de ned for time-independent elliptic PDEs [68, 95], it is generally not needed to estimate their solution.

As we will discuss in Sec. 2.3.2, PDEs with Dirichlet boundary conditions require simulating random walks that are stopped when they rst hit ∂W (Fig. 2.7, left). Hence from the perspective of stochastic processes, Dirichlet conditions model absorbing boundaries. However, a random walk need not always stop when it hits the boundary—as shown in Fig. 2.7, Neumann and Robin boundary conditions instead require random walks to re ect off ∂W in the normal direction [86, 87]. Walks eventually stop on Robin boundaries, as they are both re ecting and absorbing.

Discretized Random Walks

The standard approach for simulating a diffusion process in $\mathbb{R}^N$ is to use explicit time stepping [101, 134, 172], akin to ray marching [248] or forward Euler. The Euler-Maruyama method, for instance, uses the following update rule to integrate the SDE in Eq. 2.21 with time step $h > 0$:

$$X_{k+1} = X_k + \vec{w}(X_k)h + \sqrt{k(X_k)}\,(W_{k+1} - W_k), \quad W_{k+1} - W_k \sim N(0, h) \tag{2.23}$$

Unfortunately, this approach introduces several sources of error in bounded domains. E.g., random walks can easily leave a domain W and must be clamped to the boundary (Fig. 2.8); shrinking h reduces discretization error, but signi cantly slows down computation from needing to take many small steps inside W. Error is exacerbated in problems with variable diffusion and drift coef cients, which implicitly modify the ideal step size. Though there exist integration schemes with better convergence properties [42, 148, 155, 168], SDE integrators are fundamentally not well-suited for simulating continuous random processes in bounded domains—in Sec. 8.3.1, we will demonstrate that techniques based on WoS offer a much more favorable runtime-to-bias tradeoff for both

Figure 2.8: Discretized random walks can leave the domain, biasing results.

boundary-dominated and variable-coef cient problems, as they exactly model large steps of a Brownian random walk.

2.3.2 Stochastic Representation of PDEs

Notice that parameters k, $\overset{\#}{w}$ and s of a diffusion process resemble the coef cients of the PDEs from Sec. 2.1.1. The Feynman–Kac formula makes this relationship explicit by expressing the solution to Eq. 2.4 as an expectation over random trajectories of X_t. We refer to Øksendal [191, Sec. 8.2] for a derivation of this result (via a fundamental tool in stochastic calculus called Ito's lemma[191, Ch. 4]). Here we build up to the Feynman–Kac formula by starting with the important special case of Kakutani's principle, which effectively tells us how to solve a Laplace equation with Dirichlet boundary conditions by taking many random walks to the boundary.

Kakutani's Principle

Consider the Laplace equation

$$\begin{aligned} Du(x) &= 0 \quad \text{on } W, \\ u(x) &= g(x) \quad \text{on } \P W \end{aligned} \tag{2.24}$$

on a domain $W \quad R^N$, and let W_t be a Brownian process starting at a point $W_0 = x \; 2 \; W$. In this case, Kakutani's principle [122] states that

$$u(x) = E[g(W_t)], \tag{2.25}$$

where as before t is the (random) time when W_t rst hits the domain boundary $\P W$. In other words, the solution to a Laplace equation is just the average boundary value “seen” by random walks starting at x.

Connection To Mean Value Property. Kakutani's principle can be viewed as a generalization of the mean value property (Eq. 2.13), since in the special case where the domain W is a ball $B(x, R)$ of (any) radius R centered at the start of a random walk, the exit distribution of Brownian motion is uniform over the boundary sphere $\P B$ (i.e., it equals $1/\,j\P Bj$). Hence, Kakutani's principle in this case gives just the mean value integral

$$u(x) = E[g(W_t)] = \frac{1}{j\P B(x,R)j} \int_{\P B(x,R)} g(z)\, dz. \tag{2.26}$$

Source Term

For PDEs with a source term f, such as the Poisson equation $Du = \quad f$, the solution u additionally picks up a term capturing the average heat “felt” by a random walk along its path [191, Ch. 9]:

$$E \int_0^t f(W_t)\, dt \quad . \tag{2.27}$$

Connection To Green's function. The solution to a Poisson equation can also be expressed by convolving f with the Green's function G^W of the domain, if we assume the domain boundary $\P W$ only has zero Dirichlet conditions ($g = 0$), and $G^W = 0$ on $\P W$. In this case, the BIE in Eq. 2.9 simpli es to

$$u(x) = \int_W G^W(x,y) \; f(y)\, dy. \tag{2.28}$$

Comparing against Eq. 2.27, we thus have

$$E\int_0^t f(W_t)\,dt = \int_W G^W(x,y)\,f(y)\,dy, \tag{2.29}$$

which implies that the Green's function in fact describes the locations where a Brownian random walk is likely to spend time within the domain W [191, Sec 9.2].

Screening Term

To model the effect of absorption, as in a screened Poisson equation $\Delta u - su = f$, we incorporate the absorption coefficient $s \in \mathbb{R}_{\geq 0}$ into the boundary and source terms to get

$$E\left[e^{-st}g(W_t)\right] \quad \text{and} \quad E\left[\int_0^t e^{-st}f(W_t)\,dt\right], \tag{2.30}$$

respectively [191, Ch. 8]. Notice that larger values of s yield smaller contributions. This exponential downweighting of the solution is accounted for in the boundary integral equation for a screened Poisson equation through its Green's function and Poisson kernel (App. A.2). For a spatially varying coefficient $s(x)$, we simply replace st with $\int_0^t s(W_t)\,dt$ in both terms.

Feynman–Kac

Finally, to account for spatially varying diffusion $k(x) \in \mathbb{R}_{>0}$ and drift $\vec{w}(x) \in \mathbb{R}^N$, we replace the Brownian motion W_t with a general diffusion process X_t from Eq. 2.21. Combining expressions for the boundary, source, and absorption terms from Eqs. 2.25–2.30, we then arrive at:

The Feynman–Kac formula

For any point $x \in W$, the solution to Eq. 2.4 with Dirichlet boundary conditions satisfies

$$u(x) = E\left[e^{-\int_0^t s(X_t)\,dt}g(X_t) + \int_0^t e^{-\int_0^t s(X_s)\,ds}f(X_t)\,dt\right]. \tag{2.31}$$

With Dirichlet conditions, a random walk terminates when $X_t \in \partial W_D$. With Neumann and Robin conditions, the primary change to the Feynman–Kac formula is that X_t must be reflected off ∂W_N and ∂W_R (Fig. 2.7). We refer to Morillon [172] for details on how to augment Eq. 2.31 with contributions from non-zero Neumann and Robin data h and $\grave{}$ respectively (Eq. 2.1).

Direct Estimation Using Discretized Random Walks. To approximately compute the solution to an elliptic PDE, one can use, e.g, Euler-Maruyama with step-size h to simulate M random walks in W starting from x (Eq. 2.23). We can then estimate each walk's contribution by discretizing Eq. 2.31, and averaging results as follows:

$$\frac{1}{M}\sum_{i=1}^{M} Y_i, \quad \text{where } Y_i := e^{-h\sum_{k=0}^{N-1} s(X_k)}g(X_N) + h\sum_{k=0}^{N-1} e^{-h\sum_{l=0}^{k-1} s(X_l)}f(X_k). \tag{2.32}$$

Here N is the number of steps a random walk takes before it terminates on ∂W_D. In addition to the inefficiencies of discretized walks described in Sec. 2.3.1, a direct summation of this kind introduces further discretization error into the solution. Error also arises from naive estimation of the function $\exp\left(-\int_0^t s(X_t)\,dt\right)$, since nonlinear functions f do not in general commute with expectations ($E[f(X)] \neq f(E[X])$). We briefly turn to volume rendering next, as it will inspire techniques for solving variable-coefficient PDEs with WoS that avoid discretization entirely.

Figure 2.9: Left: The Feynman–Kac formula describes how a source term f and boundary data g contribute to the solution of an elliptic PDE with spatially varying coefficients along the trajectory of a random process X_t. Right: The volume rendering equation likewise describes the radiance $L(x, \vec{\omega})$ as a function of scattering, emission f, and the radiance g leaving the boundary, but along a straight light path rather than a random walk.

Similarity To The Volume Rendering Equation. In computer graphics, the radiative transport equation (RTE) [35] is used to describe the behavior of light in heterogeneous media that absorb, scatter and emit radiation (Fig. 2.9, left). Unlike Eq. 2.4, the RTE is only 1st order in space. It states that the radiance $L(x, \vec{\omega})$ at each point $x \in W$ and in each direction $\vec{\omega} \in \mathbb{R}^n$ satisfies

$$\begin{aligned} \vec{\omega} \cdot \nabla L(x, \vec{\omega}) - \sigma(x) L(x, \vec{\omega}) &= f(x, \vec{\omega}, L) \quad \text{on } W, \\ L(x, \vec{\omega}) &= g(x, \vec{\omega}, L) \quad \text{on } \partial W. \end{aligned} \tag{2.33}$$

This equation is recursive, since the source term $f(x, \vec{\omega}, L)$ depends on the radiance $L_s(x, \vec{\omega})$ in-scattered at x (as well as any emission $L_e(x, \vec{\omega})$); likewise, the function $g(x, \vec{\omega}, L)$ describes radiance leaving the boundary. The spatially varying extinction coefficients $\sigma(x)$ specifies the density of scattering or absorbing particles at x.

The integral representation of the RTE is called the volume rendering equation (VRE) [203, Ch. 15.1]. The VRE gives the radiance $L(x, \vec{\omega})$ as an integral along a ray $x_t := x - \vec{\omega} t$ of length d:

$$L(x, \vec{\omega}) = e^{-\int_0^d \sigma(x_t)\, dt} g(x_d, \vec{\omega}, L) + \int_0^d e^{-\int_0^t \sigma(x_s)\, ds} f(x_t, \vec{\omega}, L)\, dt. \tag{2.34}$$

It shares a close resemblance with the Feynman–Kac formula (Eq. 2.31).

Delta tracking. The VRE is typically solved using volumetric path tracing (VPT) [141], but a spatially varying $\sigma(x)$ presents challenges akin to those for the Feynman–Kac formula: approximating the transmittance function $\exp(-\int_0^d \sigma(x_t)\, dt)$ via explicit steps along x_t can yield significant error. Delta tracking [207, 270] instead rewrites Eq. 2.33 so that all spatial variation in the extinction coefficient $\sigma(x)$ is captured by a source term on the right-hand side—leaving only a constant absorption coefficient $\bar{\sigma} := \max(\sigma(x))$ [74, 139]:

$$\vec{\omega} \cdot \nabla L(x, \vec{\omega}) - \bar{\sigma} L(x, \vec{\omega}) = \underbrace{\left(f(x, \vec{\omega}, L) + (\bar{\sigma} - \sigma(x)) L(x, \vec{\omega}) \right)}_{=: f^0(x, \vec{\omega}, L)}. \tag{2.35}$$

Conceptually, fictitious null matter is added to the initially heterogeneous medium so that it has a constant density (Fig. 2.10, left). Eq. 2.35 then has the integral representation

$$L(x, \vec{\omega}) = e^{-\bar{\sigma} d} g(x_d, \vec{\omega}, L) + \int_0^d e^{-\bar{\sigma} t} f^0(x_t, \vec{\omega}, L)\, dt. \tag{2.36}$$

Figure 2.10: Left: The delta tracking method in volume rendering artificially fills a heterogeneous medium with fictitious null matter (indicated by $s_n(x)$) so that the combined density $\bar{s}$ is constant everywhere. Right: With VPT [70, Algorithm 2], we then re-weight the radiance L by $s(x)$ to account for the original heterogeneity $s(x)$, which corresponds to probabilistically sampling null-events inside the medium. Likewise in Sec. 5.2.1, our delta tracking WoS estimator will solve variable-coefficient PDEs by introducing null-events into the random walk.

This representation is more amenable to Monte Carlo integration, since the transmittance function $e^{-\bar{s}t}$ can be evaluated in closed form. Spatial variations in $s(x_t)$ are accounted for by weighting the radiance L by $\bar{s} - s(x_t)$ inside the modified source term f^0.

In Ch. 5, we will derive a generalized mean value expression for the variable-coefficient PDEs from Sec. 2.1.1 by applying the delta tracking transformation to the Feynman–Kac formula. This will lead to an integral representation amenable to walk on spheres.

Chapter 3

Basic PDE Estimators

From here on, we use the Monte Carlo method to solve the PDEs from the previous chapter, via their integral formulations. In this chapter, we first explain how we can use Monte Carlo to numerically evaluate integrals that do not have analytic solutions. We then introduce Muller [174]'s walk on spheres algorithm for simulating Brownian motion (Sec. 2.3.1) in bounded domains, which allows us to apply Monte Carlo integration to mean value expressions (Sec. 2.2) for constant-coefficient PDEs with Dirichlet boundary conditions.

3.1 Background: Monte Carlo Integration

We review elementary facts about Monte Carlo integration, and refer to Veach [251, Chapter 2] and Pharr et al. [203, Chapter 13] for a more thorough introduction. The basic idea is that an integral can be estimated by simply sampling the integrand at randomly-chosen points, and averaging results from several trials. This makes Monte Carlo easy to implement and generally applicable to a wide variety of integrands, including those containing discontinuities or singularities. It is also the only practical numerical integration technique for high-dimensional integrals, since the performance of quadrature schemes becomes exponentially worse as dimensionality increases [251, Sec. 2.2]. As a randomized algorithm, Monte Carlo gives different results depending on the random numbers used, but on average the results are statistically close to the true answer. In particular, let f be an L^1-integrable function on a domain A. Then the integral

$$I := \int_A f(x)\,dx \tag{3.1}$$

can be approximated by the Monte Carlo estimator

$$\hat{I}_N := |A| \frac{1}{N} \sum_{i=1}^{N} f(x^i), \quad x^i \sim U(A), \tag{3.2}$$

where N is any positive integer, $x^i \sim U(A)$ indicates that x^i are independent random samples drawn from the uniform distribution on A, and $|A|$ denotes A's volume. In this text, we will express all PDE estimators as single-sample estimators $\hat{I}$ (dropping the subscript $N = 1$ for brevity), with the expectation that their values will be averaged over many trials to improve accuracy.

Importantly, although $\hat{I}_N$ is called an "estimator", it does not provide merely an estimate—rather, a well-designed estimator will give the *exact* value of the integral, in expectation. More precisely, an estimator is *unbiased* if its expected value equals the true value, $E[\hat{I}_N] = I$, for any number of samples N. We quantify the accuracy of an estimator using its expected squared error $E[(\hat{I}_N - I)^2]$, which for an unbiased estimator equals its *variance*

$$V[\hat{I}_N] := E[(\hat{I}_N - E[\hat{I}_N])^2]. \tag{3.3}$$

Assuming independent samples x^i, variance goes to zero at a rate of $O(1/N)$, irrespective of the dimensionality of the integral. This means that Monte Carlo algorithms converge at the rate of

Figure 3.1: An importance sampling estimator concentrates samples where the integrand f is large (left), often by using an easy-to-sample approximation of f as its PDF p (right).

$O(1/\sqrt{N})$ to the correct result, i.e., to cut error in half, we require four times as many samples. We must therefore apply variance reduction techniques, discussed next, to reduce noise in the estimated results given a fixed sample budget. In practice, we use Welford's online algorithm [267] to compute to a numerically stable estimate of the sample variance

$$\hat{V}[\hat{I}_N] = \frac{1}{N-1}\sum_{i=1}^{N}\left(x^i - \frac{1}{N}\sum_{j=1}^{N} x^j\right)^2, \tag{3.4}$$

which is an unbiased estimator of Eq. 3.3. We say that two Monte Carlo estimators are correlated if they use the same set of independent samples to compute their respective sums.

3.1.1 Variance Reduction

Designing efficient estimators is of central importance to the adoption of Monte Carlo algorithms in downstream applications. The efficiency of an estimator $E[\hat{I}_N]$ is determined not just by its variance $V[\hat{I}_N]$, but also by the time $T[\hat{I}_N]$ needed to compute its value. Together, V and T define

$$E[\hat{I}_N] := \frac{1}{V[\hat{I}_N]\,T[\hat{I}_N]}, \tag{3.5}$$

which states that one estimator is more efficient than another if it either takes less time to produce the same variance, or if it produces less variance in the same amount of time.

Here we review variance reduction strategies from the broader Monte Carlo literature, which we will adapt to our setting in the next few chapters (but primarily in Ch. 7). This list is not exhaustive, as there likely exist other strategies for improving efficiency we do not consider here.

Importance Sampling

Let p^A be any PDF on the domain A that is nonzero on the support of the function f. Then the integral of f in Eq. 3.1 also equals the expected value of the estimator [203, Sec. 13.2]

$$\hat{I}_N^{IS} := \frac{1}{N}\sum_{i=1}^{N}\frac{f(x^i)}{p^A(x^i)}, \quad x^i \sim p^A. \tag{3.6}$$

Importance sampling refers to concentrating samples in regions where f has large values, by sampling from a density p^A that is similar to the integrand f (Fig. 3.1). The theoretical best choice is to set $p^A = cf$, where the constant of proportionality c equals 1 over the unknown value I of the integral $\int_A f(x)\,dx$ we are estimating. In this case, the expected squared error is zero, since the importance sampling estimator $\hat{I}_N^{IS}$ equals I. In practice, variance is reduced even when p^A only approximately matches f, with importance sampling becoming especially helpful if integrands are localized or have singularities. In the remainder of this text, we will use Eq. 3.6 as the default estimator for all the PDEs we consider, and drop the superscript on $\hat{I}^{IS}$ for brevity.

Control Variates

Suppose we want to estimate the integral $\int_A f(x)\,dx$, and have a function $\tilde{f}(x)$ with known integral c. A control variate strategy then is

$$\hat{I}_N^{CV} := c + \frac{1}{N}\sum_{i=1}^{N} \frac{f(x^i) - \tilde{f}(x^i)}{p^A(x^i)}, \quad x^i \sim p^A \tag{3.7}$$

i.e., estimate the difference between f and $\tilde{f}$, and shift by c. Intuitively, if $\tilde{f}$ is similar to f, then the difference is a nearly constant function that will have smaller variance than f itself (inset). This strategy will be useful for reducing noise when estimating derivatives of PDEs (Sec. 7.2).

Antithetic Variates

Antithetic variates are most useful when the integrand f is smooth and approximately linear. The basic idea is to add two estimators that use the same random samples and are negatively correlated, as the variance of the estimators together is smaller than if the samples are independent. Assume for simplicity that we want to estimate $\int_0^1 f(x)\,dx$. Then, an antithetic variate strategy is

$$\hat{I}_N^{AV} := \frac{1}{N}\sum_{i=1}^{N} \frac{f(u^i) + f(1-u^i)}{2}, \quad x^i \sim U([0,1]). \tag{3.8}$$

This estimator has zero variance when f is a linear function, and extends easily to unit hypercubes $[0,1]^s$ of dimension s using sample pairs $\{u_1^i, \ldots, u_s^i\}$ and $\{1-u_1^i, \ldots, 1-u_s^i\}$. It can also be used on more general domains via a transformation between distributions [203, Sec. 13.5]. We will apply antithetic variates in conjunction with control variates to estimate PDE derivatives.

Strati ed Sampling

Variance can also be reduced by subdividing the integration domain A into n nonoverlapping subdomains $a_1, a_2, \ldots, a_n$, such that $\bigcup_{i=1}^{n} a_i = A$. Samples are then generated in each subdomain a_i using probability densities p^{a_i}. Compared to randomly sampling over all of A, this sort of strati ed placement ensures that samples are more evenly distributed and do not clump together as much (see inset), making them less likely to miss important features of the integrand. A number of sophisticated sample placement schemes have been developed [251, Sec. 13.8] such as Latin hypercube sampling, orthogonal array sampling, and quasi-Monte Carlo (QMC) methods. QMC generally demonstrates asymptotically faster rates of convergence than standard Monte Carlo, but often at the cost of increased computation. We will stratify samples in the interior and on the boundary of a PDE domain in Sec. 7.5.

Adaptive Sampling

While importance sampling concentrates samples in regions where the integrand is large, adaptive sampling instead places samples in regions where the estimator has most variation. This

information can be computed directly from the estimated variance $\hat{V}[\hat{p}_N]$ in Eq. 3.4 to inform placement of future samples. One can then change the sampling PDF as more information about the integrand is gathered while taking more samples. Unlike importance sampling, the main disadvantage of adaptive sampling is that it can produce biased results. However, bias can be justi ed if cost is signi cantly reduced—in Sec. 7.4, we will demonstrate that this is the case in regions where the integrand is smooth.

Sample Reuse And Caching

Generating samples from a probability density p^A can be expensive. For instance, when evaluating high-dimensional integrals with random walks (as we will see in Sec. 3.2), each sample depends on a sequence of nested decisions constituting k steps of a walk. In situations where several similar integrals need to be evaluated at nearby points in a domain, one can improve ef ciency by reusing the same set of independent samples to evaluate more than one integral. Sample reuse introduces correlations between results which slow down the rate of convergence, but it can also greatly amortize the cost of generating expensive samples corresponding to, e.g, long walks. Reuse may require storing samples in a cache, before their contribution can be shared between different estimators. We will develop a caching strategy for PDEs in Sec. 7.5.

Russian Roulette And Splitting

In particle transport [26, 102], Russian roulette and splitting are closely related techniques that improve the ef ciency of an estimator by controlling the sample density in a domain. Unlike adaptive sampling, they do not introduce any bias, and ensure that each sample makes a signi cant contribution to the result.

Russian Roulette. Russian roulette provides an unbiased mechanism to discard samples that are expensive to evaluate but make little contribution to the estimator, e.g, a random walk in a highly absorbing medium (Sec. 2.3.2). Russian roulette skips these samples by replacing the single-sample estimator $\hat{p}$ with a new estimator of the form

$$\hat{p}^{RR} := \begin{cases} \hat{p}/q & \text{with probability } q, \\ 0 & \text{otherwise.} \end{cases} \tag{3.9}$$

Here the survival probability q can be determined in any number of ways. For instance, q for a killed diffusion process (Sec. 2.3.1) can be based on the value of the exponential e^{-st} in Eq. 2.30. The estimator $\hat{p}^{RR}$ then has the same expected value as $\hat{p}$:

$$E[\hat{p}^{RR}] = q \frac{1}{q} E[\hat{p}] + (1 - q) \cdot 0 = E[\hat{p}], \tag{3.10}$$

implying that it is free from bias whenever $\hat{p}$ is unbiased. Though Russian roulette does not reduce variance, it can improve ef ciency by reducing the average time spent evaluating $\hat{p}$.

Splitting. Splitting on the other hand places more samples in important or high-contribution regions of the domain. In the context of random walks, this amounts to splitting a single walk into M new independent walks, each with $1/M$'th the contribution of the original walk. Concretely, the splitting estimator replaces $\hat{p}$ with

$$\hat{p}^{split} := \frac{1}{M} \sum_{i=1}^{M} \hat{p}_i, \tag{3.11}$$

Figure 3.2: To generate samples from a probability density p, inverse transform sampling requires computing and inverting the cumulative density function P.

where the M estimates $\hat{p}_i$ are independent from $\hat{p}$ but have the same expectation. Like Russian roulette, this decomposition leads to an unbiased estimator $\hat{p}^{\text{split}}$ via linearity of expectations. Since splitting promotes better exploration of the domain, variance is reduced but at the expense of increased runtime.

To improve efficiency, we will use Russian roulette to terminate random walks in domains with absorbing media (Sec. 3.2.2) and Robin boundary conditions (Sec. 4.3.4). In Sec. 7.7, we will also combine Russian roulette and splitting into a unified weight window strategy for PDE estimators with variable coefficients.

3.1.2 Sample Generation

To evaluate any of the Monte Carlo estimators from the previous section, we need to be able to draw random samples from a probability density. Here we describe a few standard sampling procedures which we will use later to sample from Green's functions. In the next section, we then turn to walk on spheres, which at its core is a sample generation algorithm as well, but specifically for generating realizations of Brownian motion in bounded domains.

Inverse Transform Sampling

Inverse transform sampling uses the following recipe to sample from a real-valued PDF p:

1. Compute the cumulative density function $P(x) = \int_0^x p(x')\,dx'$.
2. Compute the inverse CDF $P^{-1}(y)$.
3. Generate a uniform random number $u \sim U([0,1])$.
4. Generate a random sample $x' = P^{-1}(u)$.

As shown in Fig. 3.2 (right), the intuition behind this technique is that a value in $[0,1]$ on the vertical axis maps uniquely to the probability of an "outcome" x' on the horizontal axis—outcomes chosen for a set of uniform random numbers will therefore be distributed according to the PDF p. As an example, consider the PDF $p(x) = s e^{-sx}$ of a killed diffusion process from Sec. 2.3.1, and its corresponding CDF $P(x) = 1 - e^{-sx}$. This CDF is easy to invert, $P^{-1}(y) = -\ln(1-y)/s$, and thus easy to generate samples from, $x' = -\ln(1-u)/s$ for $u \sim U([0,1])$.

For discrete outcomes, we can compute a CDF by summing up the known probability of each outcome, and using steps 3 and 4 to generate samples. The main challenge with using inverse

transform sampling on continuous PDFs is that it is not always possible to compute and invert CDFs. Rejection and weighted importance sampling relax these requirements.

Rejection Sampling

Rejection sampling draws samples from non-negative functions $f(x)$ that need not be normalized, i.e., f does not have to integrate to 1, and hence is not necessarily a PDF. Assume we have access to a PDF g that is easy to sample from (via, e.g, inverse transform sampling), and a constant c such that $f(x) \leq cg(x)$. We then repeat the following procedure till it returns a correctly distributed sample x^0:

1. Generate a uniform random number $u \sim U([0,1])$.
2. Generate a sample $x^0 \sim g(x)$.
3. Return x^0 if $u < f(x^0)/cg(x^0)$.

Intuitively, when the PDF g is uniform, more samples are rejected in regions where f is small. Tighter bounds $cg(x)$ can therefore signi cantly improve performance.

Weighted Importance Resampling

Assume we want to again sample according to an unnormalized function $f(x)$, but only have access to a stream of samples $\{x^1, x^2, \ldots\}$ generated using a PDF $g(x)$. We can choose a representative sample distributed proportionally to f by processing the stream one element at a time, and selecting—from the M samples seen so far—an x^i with probability $w(x^i)/\sum_{j=1}^{M} w(x^j)$, where the weight $w(x) := f(x)/g(x)$ [23, Alg. 3]. The next stream sample x^{M+1} replaces x^i with probability $w(x^{M+1})/\sum_{j=1}^{M+1} w(x^j)$. The stream length M need not be known ahead of time.

3.2 Walk On Spheres

Suppose we want to evaluate the solution to a basic Laplace equation $\Delta u = 0$ with Dirichlet boundary conditions g (Eq. 2.24) at some point $x_0 \in W$. The mean value property (Eq. 2.13) says that $u(x_0)$ is equal to the average of u over the boundary of any ball $B(x_0, R) \subseteq W$. Alternatively, Kakutani's principle (Eq. 2.25) says that $u(x_0)$ equals the expected value of u where continuous Brownian random walks rst hit the ball boundary ∂B. We therefore have

$$u(x_0) = \frac{1}{|\partial B(x_0, R)|} \int_{\partial B(x_0,R)} u(z)\, dz = E[u(W_t)], \tag{3.12}$$

where by symmetry a random walk W_t starting at a point x_0 is equally likely to exit through any point on ∂B—independent of how long the walk takes, or where it goes inside B (Fig. 3.3).

The mean value and stochastic perspectives both point to the same strategy for estimating $u(x_0)$: uniformly sample a point x_1 on a sphere around x_0. If x_1 is extremely close to the domain boundary (i.e., within the $\#$-shell $\partial W_D^{\#}$), grab the boundary value $g(\bar{x}_1)$ at the closest point $\bar{x}_1 \in \partial W_D$ (inset). Otherwise, evaluate $u(x_1)$. Repeated evaluation of Eq. 3.12 results in a random walk on the points $x_0 \to x_1 \to \ldots$, where

Figure 3.3: At any point x, the solution u to a Laplace problem is equal to the average boundary value reached by random walks (left), and the average value of u over a sphere $\partial B(x)$ around x (right). Both quantities can be estimated by recursively sampling points on a sphere.

for $k \geq 0$, each point x_{k+1} lies on a sphere centered at the previous point x_k. This reasoning leads to the recursive WoS estimator by Muller [174]:

$$\hat{u}(x_k) := \begin{cases} g(\bar{x}_k) & x_k \in \partial W_D^{\#}, \\ \frac{1}{|\partial B(x_k,R)|} \frac{\hat{u}(x_{k+1})}{p^{\partial B}(x_{k+1})} & \text{otherwise.} \end{cases} \tag{3.13}$$

To achieve fast convergence to ∂W_D, we draw a single point x_{k+1} from the uniform density $p^{\partial B} := 1/|\partial B|$ on the largest sphere around x_k with radius $R = \|x_k - \bar{x}_k\|$ (Fig. 1.6), which can be computed efficiently via a closest point query (Sec. 6.1). In this case, one can show that the number of steps needed to reach the $\#$-shell $\partial W_D^{\#}$ is typically $O(\log 1/\#)$ [22], and that error vanishes quickly as $\#$ is decreased (Sec. 6.3). Alg. 1 provides pseudocode.

Unlike SDE integration schemes (Sec. 2.3.1), WoS provides an exact statistical simulation of Brownian motion without introducing discretization error at every step of a random walk. Moreover, numerical quadrature is not a practical alternative to Monte Carlo here, since the recursively expanded integration domain is high-dimensional. In the remainder of this section, we describe known extensions of WoS to other constant coefficient PDEs such as the Poisson and screened Poisson equations. We also discuss estimation of derivatives, and treatment of reflecting boundary conditions.

3.2.1 Poisson Equation

We can solve a Poisson equation (Eq. 2.2) by incorporating a source term $f : W \to \mathbb{R}$ into the WoS estimator as follows:

$$\hat{u}(x_k) := \begin{cases} g(\bar{x}_k) & x_k \in \partial W_D^{\#}, \\ \frac{1}{|\partial B(x_k,R)|} \frac{\hat{u}(x_{k+1})}{p^{\partial B}(x_{k+1})} + \frac{G^{B(x_k,R)}(x_k, y_{k+1}) f(y_{k+1})}{p^B(y_{k+1})} & \text{otherwise.} \end{cases} \tag{3.14}$$

A good default strategy for the second term, shown in the inset, is to importance sample the Green's function G^B at every step of a walk by drawing a single sample $y_{k+1} \in B(x_k, R)$ from the density $p^B := G^B(x_k, y_{k+1})/|G^B(x_k)|$. Here $|G^B(x_k)|$ is the integral of G^B over B (App. A. 6). We can use either rejection sampling or inverse transform sampling to generate samples from G^B (App. A. 1.2), and can use more than one sample at each step to reduce variance if needed. We refer to Delaurentis and Romero [51, Sec. 2] for a proof that this single-sample estimator converges to the true solution $u(x)$.

ALGORITHM 1: WalkOnSpheres$(x, \#)$

Input: Starting position $x \in W$ of random walk, and $\#$-shell.
Output: Single-sample MC estimate $\hat{u}(x)$ of Poisson equation with Dirichlet conditions.

```
d, x̄ ← ClosestPt(∂W_D, x)                 ▷ Compute distance to absorbing boundary ∂W_D (Sec. 6.1)
if d < # then return g(x̄)                  ▷ Return boundary value g at closest point x̄ if x ∈ ∂W_D^#
v ← SampleUnitSphere()                     ▷ Sample direction v uniformly on unit sphere
p ← x + d v                                ▷ Set next walk position on sphere with radius d
f̂ ← SourceEstimate(x, d)                   ▷ Estimate contribution from source term f (Sec. 2.2.1)
return WalkOnSpheres(p, #) + f̂             ▷ Repeat from next walk position p
```

3.2.2 Screened Poisson Equation

To incorporate a constant absorption coefficient $s \in \mathbb{R}_0$ (Eq. 2.3), we simply replace the functions $1/|\partial B|$ and G^B in the estimator for a Poisson equation with the corresponding Poisson kernel $P^{s,B}$ and Green's function $G^{s,B}$ for a screened Poisson equation [61]; App. A. 2.2 provides explicit expressions. As discussed in Sec. 2.3.2, a nonzero s dampens the PDE solution, which can also be observed through the following relation (from Eq. A. 17) between Poisson kernels:

$$P^{s,B}(x, y) = \frac{1}{|\partial B(x, R)|} Q^{s,B}(x, y), \tag{3.15}$$

where $Q^{s,B} \in [0, 1)$ for $s > 0$. Hence, if we use the uniform density $p^B = 1/|\partial B|$ to sample the next walk position x_{k+1}, then the solution estimate $\hat{u}$ accumulates a throughput $\tilde{O}_k Q^{s,B}(x_k, x_{k+1})$ over k steps that downweights both the boundary and source contribution g and f, respectively.

We use the term throughput in analogy with the throughput of a light path in Monte Carlo rendering [203]—in the context of a screened Poisson equation, the throughput of a walk equals the probability with which a killed Brownian motion (Sec. 2.3.1) does not get absorbed in the domain W. With WoS, we can use Russian roulette (Sec.3.1.1) to realize absorptions: we terminate walks at step k with probability $1 - Q^{s,B}(x_k, x_{k+1})$, and cancel out the contribution $Q^{s,B}(x_k, x_{k+1})$ in the walks that survive. This allows us to terminate walks early, instead of waiting for them to reach $\partial W_D^\#$ while their throughput continues to shrink. In practice, Russian roulette provides large efficiency gains when solving screened Poisson equations with WoS.

3.2.3 Biharmonic Equation

Several algorithms in geometric computing require solving two or more PDEs that depend on one another, e.g, computing deformations [240] and geodesic distances [45], solving eigenvalue problems via power iterations. Here we consider the simpler example of a biharmonic equation

$$\begin{aligned} \Delta^2 u &= 0 \quad \text{on } W, \\ u &= g \quad \text{on } \partial W, \\ \Delta u &= h \quad \text{on } \partial W, \end{aligned} \tag{3.16}$$

which can be formulated as a system of two 2nd order PDEs via the substitution $v := \Delta u$:

$$\begin{aligned} \Delta u &= v \quad \text{on } W, & \Delta v &= 0 \quad \text{on } W, \\ u &= g \quad \text{on } \partial W, & v &= h \quad \text{on } \partial W. \end{aligned} \tag{3.17}$$

Figure 3.4: A comparison of the "tree walking" strategy for a biharmonic equation to a naïve nested strategy with the same number of outer walks, and 1, 10, and 100 inner walks; for this example each outer walk takes about 15 steps. The tree walking result is slightly noisier, but many times more efficient.

To evaluate u at a point $x_0 \in W$, we can apply the standard Poisson estimator (Eq. 3.14), using the Laplace estimator (Eq. 3.13) to estimate v at each step of a walk. However, this naïve strategy is quite expensive since we now have to simulate whole walks inside each step (inset, top). This requires $O(SN^2)$ steps, where N is the number of walks used for the Poisson and Laplace estimators, and each walk takes $O(S)$ steps.

To address the inefficiency above, we propose a "tree walking" strategy that re-uses partial walks to reduce cost to $O(SN)$ [221, Sec. 4.3]. In particular, at each step x_k of the "outer" walk, we still use a single point y_{k+1} to sample the source term. But rather than using the basic WoS estimator for $v(y_{k+1})$ (Eq. 3.13), we instead use the estimate $\hat{v}(y_{k+1}) = P^B(y_{k+1}, x_{k+1})h(\bar{x}_N)$, where P^B is the off-centered Poisson kernel for the ball B (Sec. 2.2.1), and x_N is the final point in the walk used to estimate $u(x_0)$. In other words, we connect a walk of length one (from y_{k+1} to x_{k+1}) to the longer walk from x_{k+1} to x_N, then use the boundary value (inset, bottom). Though the walks are now more strongly correlated due to reuse (x_{k+1} is connected to both x_k and y_{k+1}), they are also significantly cheaper to compute—in practice, we get reduced variance for equal compute time (Fig. 3.4). Similar strategies can in principle be applied to other nested sequences of PDEs.

3.2.4 Exterior Problems

The WoS estimators from the previous sections can be used to solve PDEs on the domain *exterior*, i.e., the complement of W in $\mathbb{R}^N$. Unfortunately, there is a non-zero probability for a random walk to wander off to infinity in an open domain, or in the exterior of a closed domain; this corresponds to the non-recurrent behavior of Brownian motion in dimensions greater than 1 [27, Ch. 2]. A simple strategy then is to apply Russian roulette to terminate walks that wander far from the domain boundary—examples are shown in Fig. 1.7. Nabizadeh et al. [179] instead propose an approach based on the spherical inversion of the domain via a Kelvin transformation, and demonstrate that WoS solves exterior problems with significantly lower variance under this inversion. We refer to their work for further details.

3.2.5 Spatial Derivatives

One often requires computing not just the solution to a PDE, but also its spatial derivatives for tasks involving, e.g, shape optimization [204] and gradient flows (Fig. 8.5 & 7.10). Surprisingly little has been said about estimating derivatives via WoS—one such work is by Elepov and Mikhailov [61], which briefly discusses gradients but ignores other differential operators,

higher-order derivatives, and variance reduction strategies (Sec. 7.2 & 7.5). We provide basic estimators for spatial derivatives [221, Sec. 3] derived using harmonic analysis [9]. The more general framework of Malliavin calculus [17] provides tools for also computing sensitivities with respect to quantities such as PDE coefficients (termed Greeks in financial engineering [71]).

Gradient

We can express the gradient of a harmonic function u, with respect to an evaluation point $x \in W$, via a mean value-like integral [221, App. A]

$$\nabla_x u(x) = \frac{1}{|B(x,R)|} \int_{\partial B(x,R)} u(y)\, n_y \, dy, \tag{3.18}$$

where for a d-dimensional ball of radius R, $n_y := (y - x)/R$ is the outward unit normal at y. A basic WoS estimator for the gradient then is

$$\hat{\nabla}_{x_0} u(x_0) := \frac{d}{R} u(x_1)\, n_{x_1}, \quad x_1 \sim U(\partial B(x_0,R)), \tag{3.19}$$

i.e., just uniformly sample a point x_1 on a ball around x_0, and multiply the normal n_{x_1} by the usual WoS estimate for $u(x_1)$ (Eq. 3.13). The coefficient d/R comes from the surface area to volume ratio for an d-dimensional ball. We note that estimating the gradient adds virtually no cost on top of computing the solution at x_0, apart from a multiplication with the normal. Eq. 3.19 also makes no assumptions about the boundary conditions imposed on the domain boundary ∂W—it simply requires access to an estimate of u at $x_1 \in W$. However, Eq. 3.19 does not apply when $x_0 \in \partial W_D$, since $R = 0$.

More generally, the gradient of the (off-centered) mean value property for a screened Poisson equation (Sec. 2.2.1) is given by

$$\nabla_x u(x) = \int_{\partial B(c,R)} \nabla_x P^{s,B}(x,z)\, u(z)\, dz + \int_{B(c,R)} \nabla_x G^{s,B}(x,y)\, f(y)\, dy, \tag{3.20}$$

where c is the center of the ball B. The WoS estimator for this expression requires only a single sample for the second integral at the first step of a walk. We recover the spatial gradient for a Poisson equation when $s = 0$, and the gradient for the Laplace equation in Eq. 3.18 when $f = 0$ and $c = x$. App. A gives expressions for the functions $\nabla_x P$ and $\nabla_x G$, while Ch. 7 provides variance reduction strategies for gradient estimators.

First-Order Differential Operators

Given an estimate for the gradient $\nabla_x u$, an estimate of the directional derivative $D_Z u$ along any direction $Z \in \mathbb{R}^n$ is then given by

$$D_Z u(x) = Z \cdot \nabla_x u(x). \tag{3.21}$$

All other first-order differential operators, such as divergence or curl, can be expressed via the partial derivatives $\partial u/\partial e_i = D_{e_i} u$ along the coordinate directions $e_1, \ldots, e_n$.

Figure 3.5: To simulate reflecting random walks with WoS, a standard approach [156, 160] is to offset a walk that approaches the Neumann boundary back into the domain by a fixed distance along the inward normal $\mathbf{n}$ to the boundary (left). This approach introduces discretization error into the reflecting walk simulation. Moreover, the resulting walks tend to stick to the boundary as they are attracted to it, leading to long walk lengths (right).

Higher Order Derivatives

If u is the solution to a Poisson equation (Eq. 2.2), then its Hessian $\mathbf{H}u$ can be expressed via the integral formula

$$\begin{aligned}\mathbf{H}u(\mathbf{x}) = &\frac{d^2}{R^4}\frac{1}{|\partial B(\mathbf{x},R)|}\int_{\partial B(\mathbf{x},R)} u(\mathbf{y})(\mathbf{y}-\mathbf{x})(\mathbf{y}-\mathbf{x})^T\,d\mathbf{y} \\ &-\frac{d}{R^2}\frac{1}{|\partial B(\mathbf{x},R)|}\int_{\partial B(\mathbf{x},R)} u(\mathbf{y})\,\mathbf{I}\,d\mathbf{y} \\ &+\int_{B(\mathbf{x},R)} \mathbf{H}G(\mathbf{x},\mathbf{y})\,f(\mathbf{y})\,d\mathbf{y},\end{aligned} \tag{3.22}$$

where $\mathbf{I} \in \mathbb{R}^{d\times d}$ is the identity matrix. A single-sample WoS estimator could in principle be written as

$$\hat{u}(\mathbf{x}_1) = \frac{d^2}{R^4}(\mathbf{x}_1-\mathbf{x}_0)(\mathbf{x}_1-\mathbf{x}_0)^T - \frac{d}{R^2}\mathbf{I} + |B(\mathbf{x}_0,R)|\,\mathbf{H}G(\mathbf{x}_0,\mathbf{y}_1)\,f(\mathbf{y}_1), \tag{3.23}$$

where $\mathbf{x}_1 \sim U(\partial B(\mathbf{x}_0,R))$ and $\mathbf{y}_1 \sim U(B(\mathbf{x}_0,R))$. Here we run into some difficulty as the Hessian of the Green's function $\mathbf{H}G$ involves terms that behave like Dirac deltas: without an importance sampling strategy akin to those used for point sources (Sec. 7.1), important contributions will be missed. However, we can apply integration by parts to obtain a different expression for this term, namely

$$\int_{B(\mathbf{x},R)} f(\mathbf{y})\big(\mathbf{y}(\mathbf{x},\mathbf{y})(\mathbf{y}-\mathbf{x})(\mathbf{y}-\mathbf{x})^T - f(\mathbf{x},\mathbf{y})\,\mathbf{I}\big)\,\nabla_{\mathbf{y}} f(\mathbf{y})\,\nabla_{\mathbf{x}} G(\mathbf{x},\mathbf{y})\,d\mathbf{y}, \tag{3.24}$$

where $\mathbf{y}$ and f are functions that depend on the particular choice of Green's function (see [221, App. B.2.2]), and $\nabla_{\mathbf{y}}$ denotes the gradient with respect to $\mathbf{y}$. This quantity can be estimated via a single Monte Carlo sample, though we now also need access to the gradient of the source function f. As with the gradient estimator (Sec. 3.2.5), other 2nd-order differential operators can be estimated via a Hessian estimate. See Fig. 7.4 for a numerical example and convergence plot.

3.2.6 Reflecting Boundary Conditions

So far, we have presented WoS estimators for PDEs with absorbing Dirichlet boundary conditions. For mixed boundary value problems containing reflecting Neumann conditions (Eq. 2.1), the standard approach [160] is again to terminate walks when they reach the $\#$-shell $\partial W_D^{\#}$. However, if a walk ever reaches a point $\mathbf{x}_k$ in the $\#$-shell $\partial W_N^{\#}$ around the Neumann boundary, then the Neumann data h at the closest point $\bar{\mathbf{x}}_k \in \partial W_N$ is approximated via finite differences, e.g.,

$$h(\bar{\mathbf{x}}_k) \approx \frac{u(\bar{\mathbf{x}}_k + z\mathbf{n}_{\bar{\mathbf{x}}_k}) - u(\bar{\mathbf{x}}_k)}{z}, \tag{3.25}$$

Figure 3.6: For a Poisson equation with pure Neumann conditions, local details in the PDE solution are often resolved by the first few steps of a Brownian random walk (simulated here using the walk on stars algorithm from Ch. 4), with near-constant contributions from later steps (left). However, more steps are typically needed to resolve lower frequency global details (right).

where $z > \#$ is a constant. The solution estimate at x_k is then

$$\hat{u}(x_k) := \hat{u}(\bar{x}_k + z n_{\bar{x}_k}) \quad z h(\bar{x}_k) \quad u(\bar{x}_k). \tag{3.26}$$

In other words, $z h(\bar{x}_k)$ is added to the running estimate, and the walk continues as usual from the point $\bar{x}_k + z n_{\bar{x}_k}$ obtained by nudging $\bar{x}_k$ back into the domain by a fixed distance z along the inward unit normal (Fig. 3.5, left). Mascagni and Simonov [160] call this procedure a boundary reflection; Maire and Tanré [156] and Zhou et al. [283] provide more sophisticated approximations using higher-order differences. Similar approximations are also available for Robin boundary conditions [236, 282].

Unfortunately, such reflections are often impractical for problems with a large Neumann boundary ∂W_N: the finite difference approximation introduces significant bias if z is much larger than $\#$. Yet if z is only slightly larger than $\#$, then random walks "stick" to $\partial W_N^{\#}$, and take many small steps before escaping toward the interior (Fig. 3.5, right). Fig. 8.16 (left) shows that, in practice, boundary reflections yield both slow runtime and large accumulated bias. In the next chapter, we develop a walk on stars estimator for BVPs with mixed boundary conditions [167, 224]. WoSt avoids these issues by considering larger spheres that contain the Neumann and Robin boundaries, which greatly improve both accuracy and efficiency.

Pure Neumann Conditions The solution to a Poisson equation with pure Neumann boundary conditions is determined only up to an additive constant. From the random walk perspective, there is no Dirichlet boundary to terminate on, hence contributions from the Neumann data h accumulate forever. However, shorter walks tend to resolve high-frequency details in the solution, whereas the contribution from independent longer walks is more spatially uniform (see Fig. 3.6). Based on this observation, Maire and Tanré [156] describe a WoS estimator that stops the simulation once walks become longer than a certain length, implicitly pinning an additive constant to the solution. For WoSt we will instead apply Tikhonov regularization, which makes the solution unique by adding a small absorption coefficient s to the PDE (resulting in a screened Poisson equation). In particular, we will switch to this PDE when a walk gets longer than a user-specified length (Fig. 7.5), which adds a small but controlled amount of bias. As described in Sec. 3.2.2, we can then terminate walks via Russian roulette, using a survival probability proportional to the Poisson kernel of a screened Poisson equation.

Chapter 4

Estimators For PDEs With Mixed Boundary Conditions

The original walk on spheres method solves Dirichlet problems by simulating random walks that ultimately get absorbed into the boundary. Rather than simulate many small steps of an isotropic Brownian motion (Fig. 2.7, left), this process is greatly accelerated by sampling the next point from the largest empty ball around the current walk position (Fig. 4.1, left). In this chapter, we take a basic but important step forward by developing practical strategies for incorporating Neumann and Robin boundary conditions (Eq. 2.1), which are an essential component of virtually every real physical model. To model these boundary conditions, one must additionally simulate re ecting random walks that "bounce" off the boundary (Fig. 2.7, center & right) [86, 87]. In a half space, re ecting walks amount to just taking the absolute value of Brownian motion in one coordinate direction. Hence, for polyhedral domains, a na¨ve strategy for simulating re ections is to sample the largest ball that intersects only a single boundary face, and to perform a re ection across the boundary plane if the sampled point falls outside the domain (see inset). However, the ef ciency of this strategy drops quickly as the boundary mesh is re ned.

Our strategy, which we call walk on stars (WoSt) [167, 224], is both more ef cient and more general—it is a Monte Carlo estimator for the boundary integral equation of a Laplace problem (Sec. 2.2). In short, we identify a large star-shaped region around the current walk position, and sample a point on its boundary by picking a random direction (Fig. 4.1, center & right). Like WoS, WoSt takes large steps inside the domain to quickly reach the Dirichlet boundary (these techniques are in fact equivalent for pure Dirichlet problems). Yet unlike prior WoS schemes for Neumann and Robin boundaries (Sec. 3.2.6), WoSt can also take large steps that are independent of the level of tessellation near a re ecting boundary. In Sec. 8.3, we will demonstrate that this strategy works reliably in both convex and non-convex domains without incurring large bias or variance, unlike other grid-free Monte Carlo approaches [63, 217, 235, 236, 242].

To use WoSt, the only question that must be answered is: how do we nd star-shaped regions? We propose one strategy here, using the visibility silhouette, which is easy to implement ef ciently without much overhead (Sec. 6.2). Fundamentally, however, the WoSt approach relies only on the use of star-shaped regions, and not on any particular method used to compute them, or on any particular representation of the domain boundary. Importantly, WoSt requires only few changes to an existing WoS implementation, and provides the same advantages as WoS such as progressive and output sensitive evaluation, trivial parallelization, and robustness to defective geometry, while being applicable to a broader class of problems (see, e.g, Fig. 1.9 & 4.2).

We rst develop the WoSt estimator for Neumann problems in Sec. 4.2, and then describe modi cations for Robin conditions in Sec. 4.3. We limit the discussion to Poisson equations here, and address more general linear elliptic PDEs in the next chapter. We also assume for simplicity that the domain W is a compact subset of $\mathbb{R}^N$, and provide extensions to open domains and double-sided boundary in App. B.

Figure 4.1: WoSt solves BVPs with any combination of Dirichlet, Neumann and Robin boundary conditions. Left: With a purely absorbing Dirichlet boundary ∂W_D, WoSt is equivalent to WoS, which repeatedly jumps to a random point on the largest sphere around the current walk location, and terminates when the walk enters shell $\partial W_D^{\#}$. Center: For a reflecting Neumann boundary ∂W_N, WoSt generalizes to using star-shaped regions—specifically a sphere containing a subset of ∂W_N. The next walk location is determined by intersecting a uniformly sampled ray direction against the current sphere and the visible portion of ∂W_N it contains, and picking the first hit point. Right: For a Robin boundary that is both reflecting and absorbing, the walk can additionally terminate on ∂W_R.

4.1 Star-Shaped Regions

In lieu of balls, WoSt considers regions that are star-shaped with respect to a point $x \in W$, i.e., regions whose boundary is visible from x. Though in principle any star-shaped region could be used, we use regions $St(x, R)$ given by the intersection $B(x, R) \cap W$ containing x, for a particular choice of radius R for the ball B (see Sec. 4.2.2). Similar to Eq. 2.1, we partition the region boundary into a Neumann part $\partial St_N := \partial W_N \cap \partial St$ with prescribed normal derivatives $\partial u / \partial n = h$, and a spherical part $\partial St_B := \partial B \cap \partial St$. For a Robin boundary ∂W_R, we likewise define $\partial St_R := \partial W_R \cap \partial St$ with prescribed values $\partial u / \partial n + \alpha u = \beta$.

4.2 Walk On Stars With Neumann Conditions

Letting $A := St$ and $C := B$ in Eq. 2.10, the boundary integral for a Poisson equation becomes

$$a(x)u(x) = \int_{\partial St(x,R)} P^B(x,z)u(z)\,dz + \int_{\partial St_N(x,R)} G^B(x,z)h(z)\,dz + \int_{St(x,R)} G^B(x,y)f(y)\,dy. \quad (4.1)$$

As with the mean value property used by WoS (Eq. 2.13), the solution value $u(z)$ is the only unknown in this equation: at points $z \in \partial St_N$, the normal derivative $\partial u / \partial n$ is given by the fixed Neumann data h along ∂W_N, and is not needed at points $z \in \partial St_B$ where $G^B(x, z) = 0$. Since only one quantity is unknown, estimators for this equation need not branch.

Simonov [235], and later Ermakov and Sipin [63], take a parallel approach on domains W with convex Neumann boundaries ∂W_N. In particular, they use regions formed by intersecting W with a ball B whose radius is the distance to the Dirichlet boundary, $d_{Dirichlet} := \|x - \bar{x}\|$. Hence, B can contain a subset of the Neumann boundary ∂W_N. In the convex case, such regions are automatically star-shaped. To handle arbitrary domains, WoSt instead uses visibility information to obtain star-shaped regions even near nonconvex Neumann boundaries (which in general can yield a radius $R \gg d_{Dirichlet}$), as we will see in Sec. 4.2.2.

Figure 4.2: The walk on stars method handles mixed boundary conditions, enabling it to model a richer class of problems than the original walk on spheres method. Here for instance we simulate conductive heat transfer from a toaster with Dirichlet conditions to a piece of bread with Neumann conditions, by solving a Laplace equation (top and bottom right), complementing the radiative transfer computed via ray tracing (bottom left). As with ray tracing, we can simulate directly on the full high-resolution data (bottom center) without generating a volume mesh or forming a global stiffness matrix. Since results are progressive, we can get a preview of how the toast will look faster than it takes to toast a real piece of bread (top left).

4.2.1 Monte Carlo Estimation

The Walk On Stars Estimator With Dirichlet-Neumann Conditions

A single-sample Monte Carlo estimator for Eq. 4.1 is given by

$$u(x_k) = \frac{P^B(x_k, x_{k+1})\, u(x_{k+1})}{a(x_k)\, p^{\partial St(x_k,R)}(x_{k+1})} + \frac{G^B(x_k, z_{k+1})\, h(z_{k+1})}{a(x_k)\, p^{\partial St_N(x_k,R)}(z_{k+1})} + \frac{G^B(x_k, y_{k+1})\, f(y_{k+1})}{a(x_k)\, p^{St(x_k,R)}(y_{k+1})}, \tag{4.2}$$

where for $k \quad 0$,

- the points $x_{k+1} \; 2 \; \partial St$, $z_{k+1} \; 2 \; \partial St_N$, and $y_{k+1} \; 2 \; St$ are sampled from the probability densities $p^{\partial St}$, $p^{\partial St_N}$ and p^{St} (respectively).
- R is chosen so that $S(x_k, R)$ is star-shaped.

The WoSt estimator is recursive as u appears on both sides of Eq. 4.2. Applying it iteratively leads to a random walk from one star-shaped region to another—hence the name walk on stars At a high level, each step of WoSt accumulates contributions from the (non-recursive) Neumann data h and source term f. For mixed Dirichlet-Neumann problems, the walk terminates when it enters the $\#$-shell $\partial W_D^{\#}$, using the Dirichlet data g at the closest point $\bar{x}_k \; 2 \; \partial W_D$ as the solution estimate, i.e., $u(x_k) := g(\bar{x}_k)$. For pure Dirichlet problems, WoSt reduces to WoS; for pure Neumann problems, we apply Tikhonov regularization to terminate walks at the expense of some bias (Sec. 7.3). We rst discuss how to sample the next step x_{k+1} and choose the radius R (Sec. 4.2.2), followed by sampling procedures for h and f (Sec. 4.2.3 & 4.2.4).

Figure 4.3: Left: For a ball $B(x_k, R)$ whose radius R is the distance to the Dirichlet boundary ∂W_D, the solution to a Poisson equation has to be estimated at all ray intersections, sampled proportional to signed solid angle. Center and right: WoSt instead restricts $B \setminus W$ to be star-shaped relative to x_k to avoid more than one intersection, and estimates the PDE solution at the first intersection point x_{k+1} on either ∂B inside W, or the Neumann boundary ∂W_N inside B. The radius R of a star-shaped region $St(x_k, R)$ is the minimum of the distance to the closest silhouette point on the Neumann boundary ∂W_N and the closest point on the Dirichlet boundary ∂W_D.

4.2.2 Random Walk On Star-Shaped Region

The next walk location is importance sampled from the Poisson kernel of a ball centered at the current point x_k, i.e., $x_{k+1} \sim p^{\partial St} = P^B(x_k, x_{k+1})$. For a Poisson equation in $\mathbb{R}^3$, this kernel equals

$$P^B_{3D}(x_k, x_{k+1}) = \frac{n_{x_{k+1}} \cdot (x_{k+1} - x_k)}{4\pi \|x_{k+1} - x_k\|^3}. \tag{4.3}$$

We use the same sampling density for a screened Poisson equation, as its corresponding kernel simply multiplies P^B by a constant in $[0, 1)$ determined by the absorption coefficient (Eq. A.15).

Eq. 4.3 coincides with the signed solid angle subtended by ∂St at x_{k+1} with respect to x_k [12, 114]. In rendering, this term appears in the light transport equation (LTE) [203, Eq. 14.15]. Unlike the BIE, the LTE multiplies P^B with a binary visibility $V(x, y)$ that equals 1 if x and y are mutually visible. Visibility ensures the product $V(x_k, y) P^{B(x_k,R)}(x_k, y)$ is nonnegative: positive if y is visible from x_k, and zero otherwise. Through a change of variables, this product can then be importance sampled via directional sampling i.e., cast a ray from x_k in a direction v uniformly sampled from the unit sphere, and find its first intersection with ∂St:

$$x_{k+1} := x_k + t_\partial v, \quad t_\partial := \min\{t \in [0, +\infty) : x_k + tv \in \partial St(x_k, r)\}. \tag{4.4}$$

We refer to Veach [251] for further details on the relationship between area and directional sampling. If x_k lies on the boundary, then the ray origin should be offset slightly along the inward boundary normal to avoid self-intersections, as described in Wächter and Binder [257].

Non-Visible Regions

Since Brownian motion can effectively "walk around corners" (Fig. 2.7), the BIE has no visibility term. Hence, the solution value at x_k can depend on non-visible points x_{k+1}, complicating use of directional sampling. In particular, if the region around x_k is nonconvex, then a naïve strategy is to estimate u at all intersections along a ray from x_k (Fig. 4.3, left), yielding a branching walk that increases in size exponentially. One could instead use just a single randomly selected intersection,

Figure 4.4: WoSt uses balls with radius no smaller than $\#$ to prevent walks from stopping near concave Neumann boundaries. Left: We only sample parts ∂W_N directly visible to x_k inside any ball $B(x_k, \#)$, implicitly assuming the function u is zero elsewhere. Right: Hemispherical boundary sampling ensures the next walk location x_{k+1} does not leave the domain, but also incurs a small bias b when x_k lies on a concave boundary.

but this approach yields extremely high variance (see Fig. 8.17 & 8.18), for two reasons. First, the recursive solution estimate must be multiplied by the number of intersections to account for the expected contribution from each intersection, causing a blowup in value as walk length increases. Second, the Poisson kernel alternates sign between consecutive intersection points along a ray, yielding unstable estimates due to cancellation [125, Ch. 4]. These issues are also the root cause of high variance in the walk on boundary (WoB) method [217, 242], which we compare against in Sec. 8.3.2. Moreover, using just the first intersection leads to a biased estimator, as the solution u is then effectively assumed to be zero on non-visible parts of the boundary.

Sampling Star-Shaped Regions

To avoid these issues, some past work assumes the entire Neumann boundary ∂St_N is convex [63, 235], yielding only one intersection for any region $B(x_k, R) \setminus W$ where $R := d_{\text{Dirichlet}}(x_k)$. This assumption of course limits the applicability of such estimators.

We instead let R be the minimum of the distance $d_{\text{Dirichlet}}(x_k)$ to the Dirichlet boundary, and the distance $d_{\text{silhouette}}(x_k)$ to the closest point on the visibility silhouette of ∂W_N. The connected component of $B(x_k, R) \setminus W$ containing x_k then defines a star-shaped region $St(x_k, R)$. Fig. 4.3 (center & right) shows several examples. We can thus sample points on the region boundary ∂St by simply taking the first point along a ray from x_k that intersects either $\partial B(x_k, R)$ or ∂W_N. Like the original WoS algorithm (and unlike the reflections in Fig. 3.5), WoSt can therefore take large steps when far from the Dirichlet boundary. Though other star-shaped sets could be also used, our approach is motivated by the fact that the closest silhouette point is easy to compute—as will be discussed in Sec. 6.2.1.

Epsilon Clamp For Star-Shaped Regions

Near concave parts of the Neumann boundary, the distance to the closest silhouette point on ∂W_N shrinks to zero (Fig. 4.5), stalling the progress of random walks. We hence limit the radius R used to define $St(x_k, R)$ to be greater than a user-defined $\#$, but still use only the first

Figure 4.5: The distance to the visibility silhouette shrinks as a query point approaches a concave part of ∂W_N.

ALGORITHM 2: Walk On Stars $(x, n_x, \#)$
Note: Changes to WoS from Alg. 1 are annotated with comments in blue.

Input: Starting position $x \in W$ of random walk, normal n_x at x (undefined if $x \notin \partial W_N$), and $\#$-shell.
Output: Single-sample MC estimate $\hat{u}(x)$ of Poisson equation with Dirichlet-Neumann conditions.

```
d, x̄ ← Closest_Pt(∂W_D, x)                          ▷ Compute distance to absorbing boundary ∂W_D (Sec. 6.1)
if d < # then return g(x̄)                           ▷ Return boundary value g at closest pt x̄ if x ∈ ∂W_D^#
R ← Max(Star_Region_Radius(∂W_N, x, d), #)          ▷ Compute star region St radius, # ≤ R ≤ d (Sec. 6.2.1)
v ← Sample_Unit_Sphere()                            ▷ Sample direction v uniformly on unit sphere
if x ∈ ∂W_N and n_x · v > 0 then v ← −v             ▷ Ensure v is sampled on hemisphere with axis n_x if x ∈ ∂W_N
hit, p, n_p ← Intersect(∂W_N, x, v, R)              ▷ Intersect ∂St_N with ray x + Rv, and get first hit
if not hit then p ← x + R v                         ▷ If there is no hit with ∂St_N, set next walk position on ∂St_B
ĥ_h ← Reflecting_Boundary_Estimate(x, R)            ▷ Estimate contribution from boundary term h (Sec. 4.2.3)
ĥ_f ← Source_Estimate(x, p, v, R)                   ▷ Estimate contribution from source term f (Sec. 4.2.4)
return Walk_On_Stars(p, n_p, #) + ĥ_h + ĥ_f         ▷ Repeat from next walk position p (undefined if x ∈ ∂W_N)
```

ray intersection to sample the next point x_{k+1}. This scheme incurs a small amount of bias when St is not star-shaped, since we assume the solution u is zero on any piece of ∂St not visible from x_k (Fig. 4.4, left). As with the epsilon parameter for the Dirichlet shell $\partial W_D^\#$, a smaller $\#$ value reduces bias near concave regions of ∂W_N at the expense of performance. We study this performance-bias tradeoff in Sec. 6.3. In practice our star-shaped regions tend to be much larger than $\#$, even slightly away from a concave boundary.

Hemispherical Sampling On The Neumann Boundary

When x_k lies on ∂W_N, sampling v from the unit sphere can yield points x_{k+1} outside W (inset). Here we instead sample v from the hemisphere around the normal n_{x_k}. This scheme effectively invokes the reflection principle of Brownian motion [113], across the halfplane at the base of the hemisphere. A useful consequence is that the $a(x_k) = 1/2$ in the denominator of the first term in Eq. 4.2 is canceled by the factor 1/2 we get from sampling a hemisphere, rather than a sphere. This prevents our recursive estimator from picking up a multiplicative factor of two each time a walk reaches ∂W_N. Note that if ∂W_N is concave at x_k, we again incur a small amount of bias (Fig. 4.4, right).

4.2.3 Sampling Neumann Boundary Conditions

For problems with nonzero Neumann conditions, we must evaluate the second term in Eq. 4.2. We could in theory use x_{k+1} to estimate both boundary terms, however, this approach yields biased results, as direction sampling (Eq. 4.4) never samples a point x_{k+1} on a flat Neumann boundary when $x_k \in \partial W_N$, even if h is non-zero there. This is because the Poisson kernel P^B in Eq. 4.3 is zero at points x_{k+1} where $n_{x_{k+1}} \perp (x_{k+1} - x_k)$. Moreover, even for non-flat boundaries, the ratio $G^B / p^{\partial St_N} := G^B / P^B$ in the second term in Eq. 4.2 results in high-variance estimates, as the Poisson kernel can take on both very large and small values.

Instead, we sample a point z_{k+1} uniformly on the Neumann boundary ∂W_N, and add a contribution $h(z_{k+1})$ only if z_{k+1} is also contained in ∂St_N. This estimate remains unbiased, since we effectively integrate the same function (h restricted to ∂St_N) over a larger domain. However, sampling the entire Neumann boundary can lead to high variance in the estimator, as most samples

will not lie on ∂St_N. Likewise, rejection sampling (Sec. 3.1.2) can be prohibitively expensive since ∂St_N can be much smaller than ∂W_N. In Sec. 6.2.4, we hence describe a strategy for efficiently generating visible samples z_{k+1} close to x_k, which significantly reduces variance.

4.2.4 Sampling The Source Term

Finally, we sample a point $y_{k+1} \in B(x_k, R)$ to evaluate the third term in Eq. 4.2. We reuse the ray direction v we sampled to generate x_{k+1} (Eq. 4.4), and set $y_{k+1} := x_k + t_f v$, where we sample the distance t_f proportionally to G^B (App. A. 1.2). If the sampled distance t_f is greater than $t_\partial := \|x_{k+1} - x_k\|$, then the point y_{k+1} lies outside the star-shaped region $St(x_k, R)$, and we reject it (inset). As in Sec. 4.2.2, (re)using a hemispherical direction cancels $a(x_k) = 1/2$ when $x_k \in \partial W_N$.

4.2.5 Final Estimator

Our final WoSt estimator is defined recursively as

$$\hat{u}(x_k) := \begin{cases} g(\bar{x}_k), & \bar{x}_k \in \partial W_D^{\#}, \\ \hat{u}(x_{k+1}) + \hat{p}_h + \hat{p}_f & \text{otherwise}, \end{cases} \tag{4.5}$$

where x_{k+1} is the next walk location in W or on ∂W_N, and $\hat{p}_h$ and $\hat{p}_f$ are non-recursive Neumann and source contributions, respectively. This estimator maintains the general structure of a WoS estimator, and thus introduces little implementation overhead. Alg. 2 provides pseudocode.

4.3 Walk On Stars With Robin Conditions

WoSt can take long walks when the boundary conditions are mostly Neumann: it must reflect off the Neumann boundary, and can terminate only on the Dirichlet boundary (Fig. 4.1, center). This is analogous to path tracing a scene where all materials have albedo one, such as a room of perfect mirrors. Here we extend WoSt to support Robin conditions, which provide greater physical realism in describing real-world thermal, electromagnetic, elastic and fluidic materials [86, 90, 94, 228] than purely absorbing or reflecting boundaries, corresponding to, e.g, a blackbody or perfect insulator (respectively). Monte Carlo estimators for Robin problems are also, in general, more efficient than estimators for Neumann-dominated problems, as random walks simulating partially reflecting Brownian motion [86] can be absorbed on ∂W_R (Fig. 2.7, right), whereas Neumann walks may require many steps to reach ∂W_D, resulting in high computation time.

To add support for Robin boundary to WoSt, our main modification is to change how we select the radius R of the balls $B(x, R)$ used to form star-shaped regions, leaving the rest of the algorithm largely unchanged. As shown in Fig. 4.6, the Robin coefficient m from Eq. 2.1 linearly interpolates between Neumann conditions ($m = 0$) and Dirichlet conditions ($m = \infty$), which means that as m increases, a Robin boundary ∂W_R becomes less reflecting and more absorbing. Intuitively, we expect m to have a similar interpolatory impact on R: in Fig. 4.7, we show R achieves its maximal value R_0 on a boundary when $m = 0$ (i.e., $\partial W_R \subset \partial W_N$), with B expanding freely through ∂W_N until it encounters a silhouette point. On the other hand, R equals its minimal value R_∞ when $m = \infty$ (i.e., $\partial W_R \subset \partial W_D$), as B cannot cross through ∂W_D. Otherwise, R transitions smoothly from R_0 to R_∞ as m increases, with ∂W_R becoming less "permeable" as B expands.

Figure 4.7: By increasing the coefficient m in Eq. 2.1, the radius of a star-shaped region for a Robin boundary reduces continuously from the distance to the closest silhouette point on the Neumann boundary when $m = 0$, to the distance to the closest point on the Dirichlet boundary where $m = \infty$.

In the rest of this section, we formalize this intuition for the ball size in three steps: in Sec. 4.3.1 & 4.3.2, we introduce a rectance function r_m, which adds support for Robin conditions to the BIE we use to design our WoSt estimator. In Sec. 4.3.3, we explain why the introduction of r_m necessitates selecting a radius R smaller than the one used with Neumann conditions, and how to use r_m to facilitate this selection (Fig. 4.8). Finally in Sec. 4.3.4, we show how r_m allows random walks to be terminated early through Russian roulette (Sec. 3.1.1) to improve efficiency (Fig. 4.6). Alg. 3 highlights these changes in green. Notably, this construction yields an estimator that demonstrates reliable Monte Carlo convergence with increasing number of walks for any combination of Dirichlet, Neumann and Robin conditions (Sec. 6.3). It also typically has orders of magnitude less error than other grid-free techniques for BVPs, like the walk on boundary method (Fig. 8.18).

Figure 4.6: As m increases and the boundary becomes less reflecting, Russian roulette helps make WoSt more efficient by probabilistically terminating walks on the Robin boundary, which reduces the average number of steps per walk.

4.3.1 Boundary Integral Formulation

To derive a boundary integral that accounts for Robin conditions, we follow largely the same derivation as that in Sec. 4.2. The main difference is that we use the Brakhage-Werner trick from potential theory [182] to substitute $\partial u / \partial n = \ell - m\, u$ (from the Robin condition in Eq. 2.1) on $\partial St_R := \partial W_R \setminus \partial St$. Rearranging terms then yields:

$$a(x)\, u(x) = \int_{\partial St(x,R)} r_m(x,z)\, P^B(x,z)\, u(z)\, dz + \int_{\partial St_R(x,R)} G^B(x,z)\, \ell(z)\, dz + \int_{St(x,R)} G^B(x,y)\, f(y)\, dy, \tag{4.6}$$

where the spatially-varying function r_m is de ned as

$$r_m(x,z) := \begin{cases} 1 - m(z)\frac{G^B(x,z)}{P^B(x,z)}, & \text{on } \partial St_R, \\ 1, & \text{on } \partial St_B. \end{cases} \tag{4.7}$$

Eq. 4.6 is nearly identical to Eq. 4.1 for Neumann problems, and even reduces to it when $m = 0$. We will treat Neumann conditions as special case of Robin moving forward. Importantly, u is the only (recursively-de ned) unknown in this equation. Similar to Sec. 4.2.1, we can therefore use single-sample Monte Carlo to derive a recursive WoSt estimator with Robin conditions.

4.3.2 Monte Carlo Estimation

The Walk On Stars Estimator With Robin Conditions

A single-sample Monte Carlo estimator for Eq. 4.6 is given by

$$\hat{u}(x_k) = \frac{r_m(x_k, x_{k+1})\, P^B(x_k, x_{k+1})\, \hat{u}(x_{k+1})}{a(x_k)\, p^{\partial St(x_k,R)}(x_{k+1})} + \frac{G^B(x_k, z_{k+1})\, \grave{\ }(z_{k+1})}{a(x_k)\, p^{\partial St_R(x_k,R)}(z_{k+1})} + \frac{G^B(x_k, y_{k+1})\, f(y_{k+1})}{a(x_k)\, p^{St(x_k,R)}(y_{k+1})}. \tag{4.8}$$

Given how similar the integrands are in Eq. 4.1 and 4.6, we can sample the points $x_{k+1} \in \partial St$, $z_{k+1} \in \partial St_R$, $y_{k+1} \in St$ using the same densities $p^{\partial St}$, $p^{\partial St_R}$, p^{St} (respectively) as those in Sec.4.2.1–4.2.4. Apart from the introduction of r_m, the WoSt estimator with Robin conditions is unchanged from the estimator with Neumann conditions. Next, we show how to select a radius R for each star-shaped region $St(x_k, R)$, and terminate random walks on ∂W_R—these are the only two places where estimation deviates from Sec. 4.2 (see Alg. 3). We treat Dirichlet conditions as separate from Robin for algorithmic convenience, even though they are a special case with $m = \infty$.

4.3.3 Using Re ectance To Select Ball Radius

As a rst attempt, we could choose R to equal the distance $d_{silhouette}$ to the closest silhouette point on ∂W_R from x, i.e., the radius R_0 we use for Neumann problems (Fig. 4.8, center). To understand why this is a bad choice for R, we must consider the values $r_m(x,z)$ (Eq. 4.7) assumes on different parts of $\partial St(x,R)$. In particular, irrespective of m, $r_m(x,z) = 1$ at points $z \in \partial St_B$, since $G^B(x,z) = 0$. When $m = 0$, $r_m(x,z)$ likewise simpli es to 1 at points $z \in \partial St_R$, recovering the original setup for Neumann problems. However, when $m > 0$, r_m in general varies between $-\infty$ and 1 if we use $R = R_0 - d_{silhouette}(x)$. This choice of radius then leads to extremely high variance in the recursive estimator in Eq. 4.8, for two reasons. First, the solution estimate $\hat{u}$ accumulates a throughput[1] $\tilde{O}_k\, r_m(x_k, x_{k+1})$ that becomes unbounded in magnitude as walk length k increases. Second, the function r_m, and thus throughput, can change sign along a walk, which results in unstable estimates due to cancellations [125, Ch. 4]. In App. C, we provide an operator-theoretic analysis of boundary integral equations to more formally explain the issues with na¨ve estimation of Eq. 4.6 and our solution to it, which we discuss below.

[1]Unlike the throughput of an estimator for a screened Poisson equation (Sec. 3.2.2), the throughput of a walk in this setting represents the probability with which Brownian motion is re ected off (and hence not absorbed on) ∂W_R.

Figure 4.8: We use re ectance values r_m on the Robin boundary ∂W_R to help determine the radius R of a star-shaped region $St(x_k, R)$. Left: For a ball $B(x_k, R)$ where R is the distance to the Dirichlet boundary ∂W_D, the PDE solution must be estimated at all ray intersections sampled proportional to signed solid angle. Center: WoSt with Neumann conditions avoids multiple intersections by instead restricting W to be star-shaped relative to x_k, and estimating the solution at a single intersection point $x_{k+1} \in \partial St$. Right: For Robin conditions with $m > 0$, R is restricted further to ensure that r_m has a value between 0 and 1.

Shrinking The Radius

To ensure throughput remains positive and bounded regardless of walk length, we choose a radius $R \leq R_0$ such that $r_m \in [0, 1]$. To achieve this, we substitute expressions for the 3D Green's function and Poisson kernel of a ball $B(x, R)$ (App. A. 1.2) into Eq. 4.7. For any point $z \in \partial St_R$,

$$r_m(x, z) = 1 - \frac{m(z)\, r}{\cos\theta} \left(1 - \frac{r}{R}\right), \tag{4.9}$$

where $r = \|z - x\|$ and $\cos\theta = n_z \cdot (z - x)/r$. The terms $1 - r/R$ and $\cos\theta$ are both positive, since $r \leq R$ and ∂St_R is front-facing by construction (as St is star-shaped). To restrict $r_m \in [0, 1]$, we therefore require

$$\frac{m(z)\, r}{\cos\theta} \left(1 - \frac{r}{R}\right) \leq 1. \tag{4.10}$$

Rearranging terms then gives an upper bound on the radius R,

$$R \leq \frac{r}{1 - \frac{\cos\theta}{m(z)\, r}} \quad \text{when } r > \frac{\cos\theta}{m(z)}, \tag{4.11}$$

which must hold at all points $z \in \partial St_R$ (Fig. 4.8, right). When $r \leq \cos\theta / m(z)$, $r_m \in [0, 1]$ for any $r < R$; in this case we set R equal to the distance $d_{Dirichlet}(x)$ to ∂W_D, or ∞ when $\partial W_D = \varnothing$.

As m increases from 0 to ∞ on ∂W_R, the bound in Eq. 4.11 reduces continuously from $R_0 = d_{silhouette}(x)$ to $R_\infty = d_{Dirichlet}(x)$. It asymptotically recovers the radii R_0 and R_∞ WoSt uses for pure Neumann and Dirichlet conditions as m approaches 0 and ∞ (respectively). With this bound for the star-shaped region radius R, we call r_m the re ectance for the Robin boundary inside $St(x, R)$, as it encodes the probability with which a random walk is re ected off ∂St_R. In Sec. 6.2.2, we describe how to compute R ef ciently on triangle meshes, with only small modi cations to how $d_{silhouette}$ is computed for ∂W_N (Sec. 6.2.1). App. D provides corresponding expressions for the re ectance and radius bound in 2D domains.

ALGORITHM 3: Walk On Stars $(x, n_x, \#)$

Note: Changes to WoSt from Alg. 2 are annotated with comments in green.

Input: Starting position $x \in W$ of random walk, normal n_x at x (undefined if $x \in \partial W_R$), and # shell.

Output: Single-sample MC estimate $\hat{u}(x)$ of Poisson equation with Dirichlet-Robin conditions.

1: $d, \bar{x} \leftarrow$ Closest Pt $(\partial W_D, x)$. Compute distance to absorbing boundary ∂W_D (Sec. 6.1)
2: if $d < \#$ then return $g(\bar{x})$. Return boundary value g at closest point $\bar{x}$ if $x \in \partial W_D^{\#}$
3: $R \leftarrow$ Max(Star Region Radius $(\partial W_R, x, d), \#)$. Compute star region St radius, $\# \leq R \leq d$ (Sec. 6.2.2)
4: $v \leftarrow$ Sample Unit Sphere () . Sample direction v uniformly on unit sphere
5: if $x \in \partial W_R$ and $n_x \cdot v > 0$ then $v \leftarrow -v$. Ensure v is sampled on hemisphere with axis n_x if $x \in \partial W_R$
6: hit, $p, n_p \leftarrow$ Intersect $(\partial W_R, x, v, R)$. Intersect ∂St_R with ray $x + Rv$, and get first hit
7: if not hit then $p \leftarrow x + R v$. If there is no hit with ∂St_R, set next walk position on ∂St_B
8: $\hat{u}_r \leftarrow$ Reflecting Boundary Estimate (x, R) . Estimate contribution from boundary term $\hat{u}_r$ (Sec. 4.2.3)
9: $\hat{u}_f \leftarrow$ Source Estimate (x, p, v, R) . Estimate contribution from source term f (Sec. 4.2.4)
10: $r_m \leftarrow$ Clamp $(1 - m(p) G^{B(x,R)}(x, p) / P^{B(x,R)}(x, p), 0, 1)$. Compute and clamp rectance $r_m \in [0, 1]$
11: if $r_m <$ Sample Uniform $(0, 1)$ then return $\hat{u}_r + \hat{u}_f$. Attempt to terminate walk using Russian roulette
12: return Walk On Stars $(p, n_p, \#) + \hat{u}_r + \hat{u}_f$. Repeat from next walk position p (undefined if $x \in \partial W_R$)

Epsilon Clamp

Irrespective of the value of m, the radius R of a star-shaped region $St(x_k, R)$ shrinks to zero as x_k approaches concave parts of ∂W_R (Fig. 4.5). As in Sec. 4.2.2, we clamp $R \geq \#$ to prevent walks from stalling near ∂W_R. As $St(x_k, \#)$ may no longer be star-shaped (Fig. 4.4), we additionally clamp r_m to $[0, 1]$ to ensure throughput remains bounded (Alg. 3, line 10). This clamping is justified by the fact that as we make $\#$ (and thus St) smaller, the value of r_m automatically approaches 1. In Fig. 6.6, we use different $\#$ values to empirically study the impact clamping has on bias and performance.

4.3.4 Using Russian Roulette To Terminate Walks

Using direction sampling (Eq. 4.4) to select the next walk location x_{k+1} perfectly importance samples (and hence cancels out) the Poisson kernel $P^B(x_k, x_{k+1})$ in Eq. 4.8, but not the rectance $r_m(x_k, x_{k+1})$. As our choice of R guarantees $r_m(x_k, x_{k+1}) \in [0, 1]$, we can also cancel out this term using Russian roulette: we terminate walks at step k with probability $1 - r_m(x_k, x_{k+1})$, and cancel out the contribution $r_m(x_k, x_{k+1})$ in the walks that survive (Alg. 3, line 11). Using Russian roulette allows us to maintain a constant throughput of 1 in our walks and terminate them early, instead of waiting for walks to reach $\partial W_D^{\#}$ while their throughput continues to shrink. This often leads to large efficiency gains, as we show in Fig. 4.6. We note that Russian roulette is not possible with Neumann conditions, where rectance always equals 1. In this case, a walk must continue until it reaches $\partial W_D^{\#}$. Otherwise it never terminates when $\partial W_D = \varnothing$, which necessitates using Tikhonov regularization (Sec. 3.2.6).

Chapter 5

Estimators For PDEs With Spatially Varying Coefficients

In this chapter, we directly resolve detailed effects of spatially varying material densities (inset)—without resorting to homogenization of PDE coefficients (Fig. 5.1). WoS has previously been applied to a limited set of problems with piecewise constant coefficients [148, 155]; ours is the first grid-free method [223] for fairly general continuously-varying coefficients. Though the Monte Carlo estimators we have developed till now cannot handle the variable coefficients $k(x)$, $\overset{\#}{w}(x)$ and $s(x)$ from Sec. 2.1.1, they can solve PDEs with a variable source term $f(x)$. We hence apply a series of transformations (Fig. 5.2) that convert a variable-coefficient PDE such as Eq. 2.4 into an equivalent constant-coefficient screened Poisson equation (Eq. 2.3); we then use the integral version of screened Poisson (Eq. 2.15) to design modified walk on spheres algorithms. From a stochastic perspective, these transformations are equivalent to writing the Feynman–Kac formula purely in terms of Brownian motion, rather than a generic diffusion process (Sec. 2.3).

Our method applies whenever the PDE

$$\begin{aligned} \nabla \cdot (k(x)\nabla u(x)) - s(x)u(x) &= f(x) \quad \text{on } W, \\ u(x) &= g(x) \quad \text{on } \partial W_D \end{aligned} \tag{5.1}$$

is elliptic—which holds if the diffusion coefficient $k(x)$ is strictly positive and the absorption coefficient $s(x)$ is nonnegative [64, 72]. For brevity, we omit the drift term $\overset{\#}{w}(x) \cdot \nabla u(x)$ in Eq. 2.4 here, though the approach remains unchanged for equations with drift (see App. E). We focus on Dirichlet boundary conditions, and discuss extensions to reflecting conditions in Sec. 5.3.2. For readers not interested in the derivation, Eq. 5.9 gives the final integral formulation of Eq. 5.1 and Sec. 5.2 discusses algorithms, which we will evaluate in Ch. 8.

5.1 Transformations

Second order. We first expand the 2nd order term $\nabla \cdot (k(x)\nabla u)$ in Eq. 5.1 via the product rule. We then divide the resulting equation by $k(x)$, and apply the identity $\nabla \ln(k(x)) = \nabla k(x)/k(x)$ to get

$$\Delta u(x) + \nabla \ln(k(x)) \cdot \nabla u(x) - \frac{s(x)}{k(x)} u(x) = -\frac{f(x)}{k(x)}. \tag{5.2}$$

At this point the 2nd order term Δu no longer has variable coefficients, but spatial variation in the lower order terms remains.

Figure 5.1: Distribution of heat (inset) radiating from in nitely many blackbodies—about 600M effective boundary vertices are visible from this viewpoint alone (we visualize a 2D slice of the full 3D solution). Our Monte Carlo PDE solver (Sec. 5.2.1) directly captures ne geometric detail and intricate spatially varying coef cients without volumetric meshing, sampling, or homogenizing the 3D domain, by building on techniques from volume rendering.

First order. A Girsanov transformation re-expresses a random process under a change of probability measure, e.g, from a generic diffusion process X_t to an ordinary Brownian motion W_t [191, Ch. 8]. As shown in App. E, applying this transformation to Eq. 5.2 eliminates the 1st order operator from Eq. 5.2, shifting all spatial variation into the 0th order term:

$$\begin{aligned} \Delta U(x) - s^0(x)U(x) &= f^0(x) \quad \text{on } W, \\ U(x) &= g^0(x) \quad \text{on } \partial W_D. \end{aligned} \tag{5.3}$$

Here,

$$U(x) := \sqrt{k(x)}\, u(x), \quad g^0(x) := \sqrt{k(x)}\, g(x), \quad f^0(x) := \frac{\sqrt{k(x)}}{k(x)}\, f(x),$$

$$\text{and} \quad s^0(x) := \frac{s(x)}{k(x)} + \frac{1}{2}\frac{\nabla k(x)}{k(x)} \cdot \frac{\nabla \ln(k(x))}{2}.$$

Equation 5.3 is equivalent to our original PDE with variable coef cients in Eq. 5.1, which can be veri ed by substituting the expressions for U, g^0, f^0 and s^0 back into this equation.

Unlike the Feynman–Kac formula in Eq. 2.31 which involves a diffusion process X_t, the stochastic formula for Eq. 5.3 uses only simple Brownian motion W_t:

$$U(x) = E\left[e^{-\int_0^t s^0(W_t)\,dt} g^0(W_t) + \int_0^t e^{-\int_0^t s^0(W_s)\,ds} f^0(W_t)\,dt \right]. \tag{5.4}$$

Zeroth Order. The only remaining term on the left-hand side of Eq. 5.3 with a spatially varying coef cient is the 0th order screening term $s^0(x)U$. We hence apply a transformation inspired by delta tracking (Eq. 2.35) to shift this heterogeneity to a source term on the right-hand side. In doing so, we introduce a coef cient $\bar{s} > 0$ by subtracting $\bar{s}U$ from both sides of Eq. 5.3. The result is a PDE with the same basic form as a screened Poisson equation:

$$\begin{aligned} \Delta U(x) - \bar{s}U(x) &= f^0(x, U) \quad \text{on } W, \\ U(x) &= g^0(x) \quad \text{on } \partial W_D, \end{aligned} \tag{5.5}$$

Figure 5.2: An overview of the transformations we apply to the variable-coefficient PDE in Eq. 5.1 to derive an integral formulation amenable to Monte Carlo estimation with WoS.

where

$$f^0(x, U) := f(x) + (\bar{s} - s^0(x))\,U(x). \tag{5.6}$$

Though only constant coefficients now appear on the left-hand side, no approximation of any kind has been introduced. Unlike a typical linear PDE however, the solution U appears on the right-hand side. As in volume rendering, we will account for this recursive dependence by applying recursive Monte Carlo estimation (Sec. 5.1.2)—a strategy not available in the traditional setting of, e.g., finite element methods.

Like the transformed VRE in Eq. 2.36 and the stochastic formulas in Eq. 2.30, the stochastic expression for Eq. 5.5 also has a transmittance function $e^{-\bar{s}t}$ that no longer varies spatially:

$$U(x) = E\left[e^{-\bar{s}t} g^0(W_t) + \int_0^t e^{-\bar{s}t} f^0(W_t, U)\, dt \right]. \tag{5.7}$$

5.1.1 Integral Representation

We can now express the solution to Eq. 5.5 using the integral form of the constant coefficient screened Poisson equation (Eq. 2.15):

$$U(x) = \int_{\partial B(c,R)} P^{\bar{s},B}(x,z)\, U(z)\, dz + \int_{B(c,R)} G^{\bar{s},B}(x,y)\, f^0(y, U)\, dy. \tag{5.8}$$

Recall that x can be any point inside the ball $B(c,R)$—not just its center. Finally, we make the substitution $U(x) = \sqrt{k(x)}\, u(x)$ from Eq. 5.3 to write this integral in terms of the original function u:

$$u(x) = \frac{1}{\sqrt{k(x)}} \left[\int_{\partial B(c,R)} P^{\bar{s},B}(x,z) \sqrt{k(z)}\, u(z)\, dz + \int_{B(c,R)} G^{\bar{s},B}(x,y)\, f^0(y, \sqrt{\bar{k}}\, u)\, dy \right]. \tag{5.9}$$

Unlike Eq. 2.15, we now have an integral equation that has unknown solution values u both in its boundary and volume terms. We will evaluate this integral via recursive application of Monte Carlo integration.

5.1.2 Monte Carlo Estimator

A single-sample estimator for Eq. 5.9 at a point $x_k \in B(c,R)$ is given by

$$\hat{u}(x_k) := \frac{1}{\sqrt{k(x_k)}} \left(\underbrace{\frac{P^{\bar{s},B}(x_k, x_{k+1})}{p^{\partial B(c,R)}(x_{k+1})} \sqrt{k(x_{k+1})}\, \hat{u}(x_{k+1})}_{\text{evaluate boundary term with probability } P^{\partial B}} + \underbrace{\frac{G^{\bar{s},B}(x_k, y_{k+1})\, f^0(y_{k+1}, \sqrt{\bar{k}}\, \hat{u})}{p^{B(c,R)}(y_{k+1})}}_{\text{evaluate volume term with probability } P^{B}} \right), \tag{5.10}$$

Figure 5.3: Left: Unlike WoS for constant coefficient problems, the delta tracking variant jumps to a random point either inside or on the boundary of the largest ball $B(x_k)$ centered at x_k. As in volume rendering (center left), null-events sampled inside B re-weight the solution estimate to account for spatial variations in the PDE coefficients. Center right: The next flight variant of WoS resolves variability in PDE coefficients by evaluating off-centered versions of the Green's function and Poisson kernel of a $B(x_k)$ in a random walk. Any walk inside B terminates at a point $x^0_{k+1} \in \P B(x^0_k)$ to avoid branching, similar to the next flight method for volume rendering which estimates transmittance along a ray with a predetermined endpoint (right).

where x_{k+1} and y_{k+1} are points sampled on the surface of and inside $B(c, R)$ according to probability densities $p^{\P B}$ and p^B (respectively). Values $P^{\P B}, P^B \in (0, 1]$ control the probability of sampling the boundary and volume terms. Letting $P^{\P B} = P^B = 1$ yields exponential growth in the number of steps, since each walk branches into two (see inset), and walks do not terminate until both x_{k+1} and y_{k+1} are contained in $\P W^{\#}_D$. In the next section, we develop two WoS algorithms that avoid branching via a careful choice of $P^{\P B}$ and P^B.

5.2 Algorithms

Due to the diversity of heterogeneous phenomena in nature, different algorithms for solving the volume rendering equation (Eq. 2.34) adopt different strategies to trade off between variance, bias, and computational cost [188]. Likewise, an algorithm for solving elliptic PDEs will be more effective when it is well-matched to the way coefficients are distributed in space. We provide a unified framework, based on Eq. 5.10, which enables us to explore variants of WoS appropriate for different problems—akin to the unidirectional estimator in Georgiev et al. [77, Eq. 14]. In particular, we devise two estimators inspired by the delta tracking [207, 270] and next flight [44] methods from volume rendering (Fig. 5.3). We also describe how to estimate the spatial gradient of the solution to Eq. 5.1 in Sec. 5.3.1.

5.2.1 Delta Tracking Variant Of Walk On Spheres

To avoid branching, our delta tracking variant of WoS uses a special property of the Poisson kernel $P^{\bar{s},B}$ of a screened Poisson equation when x_k is at the ball center. Assuming $\bar{s} > 0$, and letting $|G^{\bar{s},B}(x)|$ be the integral of $G^{\bar{s},B}$ over $B(x, R)$ (Eq. A.14), we have

$$P^{\bar{s},B}(x_k, x_{k+1}) = \frac{1 - \bar{s}|G^{\bar{s},B}(x_k)|}{|\P B(x_k, R)|}. \tag{5.11}$$

Since $\bar{s}|G^{\bar{s}}(x_k)| \in (0, 1)$ (see Eq. A.18), we can sample the boundary and volume terms with probabilities $P^{\P B} := 1 - \bar{s}|G^{\bar{s},B}(x_k)|$ and $P^B := 1 - P^{\P B}$, yielding a non-branching estimator:

ALGORITHM 4: Delta Tracking WoS$(x, \#)$

Input: Starting position $x \in W$ of random walk, and $\#$-shell.
Output: Single-sample MC estimate $\hat{u}(x)$ of Eq. 5.1 with Dirichlet conditions.

```
d, x̄ ← Closest Pt(∂W_D, x)                          ▷ Compute distance to absorbing boundary ∂W_D (Sec. 6.1)
if d < # then return g(x̄)                            ▷ Return boundary value g at closest point x̄ if x ∈ ∂W_D^#
y ← G^{s̄,B}(x, y) / |G^{s̄,B}(x)|                     ▷ Sample point y from G^B inside ball B(x, d) (Sec. A.2.2)
f̂ ← |G^{s̄,B}(x)| f(y) / √(k(x)k(y))                  ▷ Estimate contribution from source term f (Eq. 5.3)
if Sample Uniform(0, 1) < s̄|G^{s̄,B}(x)| then          ▷ Sample volume term
    return √(k(y)/k(x)) (1 − s⁰(y)/s̄) Delta Tracking WoS(y, #) + f̂      ▷ Repeat from next walk position y
else                                                  ▷ Sample boundary term
    z ← 1/|∂B(x, d)|                                  ▷ Sample point z uniformly on ∂B(x, d)
    return √(k(z)/k(x)) Delta Tracking WoS(z, #) + f̂                   ▷ Repeat from next walk position z
```

The Delta Tracking Estimator

$$\hat{u}(x_k) := \begin{cases} g(\bar{x}_k), & \bar{x}_k \in \partial W_D^{\#}, \\ \dfrac{1}{\bar{s}} \dfrac{1}{\sqrt{k(x_k)}} f^0(y_{k+1}, \sqrt{\bar{k}}\,\hat{u}), & u \sim U(0,1) \le \bar{s}|G^{\bar{s},B}(x_k)|, \\ \dfrac{\sqrt{k(x_{k+1})/k(x_k)}}{\bar{s}}\,\hat{u}(x_{k+1}) & \text{otherwise.} \end{cases} \tag{5.12}$$

This estimator importance samples x_{k+1} and y_{k+1} via the densities $p^{\partial B} := 1/|\partial B(x_k, R)|$ and $p^B := G^{\bar{s},B}(x_k, y_{k+1})/|G^{\bar{s},B}(x_k)|$, respectively. Use of constant-coefficient kernels $P^{\bar{s},B}$ and $G^{\bar{s},B}$ is critical, as kernels for the varying coefficient $s^0(x)$ from Eq. 5.3 are not known in closed form. However, spatial variation in $s^0(x)$ is still accounted for by the recursive source term f^0 (Eq. 5.6), which corresponds to probabilistically sampling null-events (Fig. 5.3, left). Alg. 4 provides pseudocode.

The coefficient $\bar{s}$ is the only free parameter in this algorithm, and must be strictly positive to ensure that $P^B > 0$. In volume rendering one typically lets $\bar{s} = \max(s(x))$, which enables closed-form sampling of volumetric events (absorption, scattering, or null scattering) and boundary reflections. We instead let $\bar{s} = \max(s^0(x)) - \min(s^0(x))$, since in general $s^0(x)$ can have both positive and negative values at different points $x \in W$. More recent volume rendering research treats $\bar{s}$ as a control variate rather than a bound [77, 188], to reduce variance based on the profile of the coefficients. In conjunction with clever choices for $p^{\partial B}$ and p^B, these strategies can be more efficient than delta tracking; we leave such extensions to future work.

5.2.2 Next Flight Variant Of Walk On Spheres

The delta tracking variant of WoS takes more steps as $\bar{s}$ increases (Fig. 5.4, left), since the Green's function becomes more localized (Fig. 2.4) and it becomes more likely that we sample a point in the volume than on the boundary ($P^B > P^{\partial B}$). Longer walks are ultimately more expensive, as distance queries are usually the bottleneck for WoS (much like ray intersections in path tracing).

We hence propose a variant based on the *next flight* scheme of Cramer [44], which takes big steps even when $\bar{s}$ is large (Fig. 5.3, right). As usual we walk along points $x_0^0, x_1^0, \ldots$ sampled from successive spheres $\partial B(x_{k-1}, R)$, always estimating both boundary and volume terms ($P^{\partial B} = P^B = 1$). But rather than start a new walk to the boundary for the volume term, we take a "short

Figure 5.4: Left: As $\bar{s}$ increases, the delta tracking variant of WoS requires more distance queries which reduces run-time performance, while the number of queries for the next flight strategy remain unchanged. Right: For smaller coefficients $s(x)$, next flight exhibits higher variance due to greater correlation among samples. For larger coefficients $s(x) + 10$, delta tracking exhibits more variance at equal compute time due to more distance queries.

off-center walk" $x_k^1, x_k^2, \ldots, x_k^M$ within each ball $B(x_k^0, R)$, and re-use the estimate of the boundary contribution at x_{k+1}^0 for all steps in this short walk. An expression for this estimator is obtained by recursively expanding the definition of $\hat{u}$ in the volume term of Eq. 5.10:

The Next Flight Estimator

$$\hat{u}(x_k^0) := \frac{1}{\sqrt{k(x_k^0)}} \sqrt{k(x_{k+1}^0)}\, \hat{u}(x_{k+1}^0)\, \hat{p}(x_k^0, x_{k+1}^0) + \hat{b}_f(x_k^0),$$

$$\text{where} \quad \hat{p}(x_k^0, z) := \sum_{j=0}^{M} \frac{P^{\bar{s},B}(x_k^j, z)}{p^{\neg B}(z)} \prod_{l=0}^{j-1} W(l),$$

$$\hat{b}_f(x_k^0) := \sum_{j=1}^{M} \frac{f(x_k^j)}{\sqrt{k(x_k^j)}\,(\bar{s} - s^0(x_k^j))} \prod_{l=0}^{j-1} W(l),$$

$$W(l) := \frac{G^{\bar{s},B}(x_k^l, x_k^{l+1})\,(\bar{s} - s^0(x_k^{l+1}))}{p^B(x_k^{l+1})}. \tag{5.13}$$

For each point x_k^l, the subscript k indexes steps in the walk; the superscript l indexes points along the short walk in $B(x_k^0, R)$. The number of terms M is determined by using the throughput $\prod_{l=0}^{j-1} W(l)$ as a probability for Russian roulette. Parameters $\bar{s}$, p^B and $p^{\neg B}$ are the same as in Sec. 5.2.1, but the Green's function $G^{\bar{s},B}$ and Poisson kernel $P^{\bar{s},B}$ must now be either evaluated or sampled using general off-centered formulas (App. A. 2.2). Alg. 5 provides pseudocode.

A key benefit of next flight is that additional distance queries are not needed to evaluate $\hat{p}$ and $\hat{b}_f$ within $B(x_k, R)$. However, decreased computation comes at the cost of increased correlation and variance in $\hat{p}$ from reuse of $\hat{u}(x_{k+1})$ and evaluation of off-centered kernels; see Fig. 5.4 (right).

5.3 Extensions

5.3.1 Spatial Gradient

Applications often require computing not just the solution to a PDE, but also the spatial gradient of the solution. Fortunately, estimating the gradient of Eq. 5.9 adds virtually no cost on top of estimating the solution itself. In particular, either of our variable-coefficient WoS algorithms can

ALGORITHM 5: NextFlight WoS$(x, \#)$

Input: Starting position $x \in W$ of random walk, and $\#$-shell.
Output: Single-sample MC estimate $\hat{u}(x)$ of Eq. 5.1 with Dirichlet conditions.

```
d, x̄ ← ClosestPt(∂W_D, x)                    ▷ Compute distance to absorbing boundary ∂W_D (Sec. 6.1)
if d < # then return g(x̄)                    ▷ Return boundary value g at closest pt if x ∈ ∂W_D^#
p̂ ← 0, b̂_f ← 0, W ← 1                        ▷ Initialize series from Eq. 5.13
z ← 1/|∂B(x,d)|                              ▷ Sample point z uniformly on ball boundary ∂B(x,d)
x^c ← x                                      ▷ Initialize temporary variable x^c to track walk position
while True do                                ▷ Perform "off-center walk" inside B
    p̂ ← p̂ + |∂B(x,d)| P^{s̄,B}(x^c, z) W      ▷ Accumulate boundary contribution
    P^RR ← min(1, W)                         ▷ Compute survival probability (Sec. 3.1.1)
    if P^RR < SampleUniform(0,1) then break  ▷ Attempt to terminate walk using Russian roulette
    W ← W / P^RR                             ▷ Update path throughput with survival probability
    x^l ← 1/|B(x,d)|                         ▷ Sample next walk position x^l inside B
    W ← |B(x,d)| G^{s̄,B}(x^c, x^l) (s̄ − s^0(x^l)) W   ▷ Update path throughput
    b̂_f ← b̂_f + f(x^l) W / √k(x^l) (s̄ − s^0(x^l))     ▷ Accumulate source contribution
    x^c ← x^l                                ▷ Update current walk position
return √(1/k(x)) √k(z) NextFlight WoS(z, #) + p̂ + b̂_f   ▷ Repeat from next walk position z
```

be used to evaluate the following integral expression for $\nabla_x u(x)$ in a ball $B(x, R)$:

$$\nabla_x u(x) = \sqrt{\frac{1}{k(x)}} \int_{\partial B(c,R)} \nabla_x P^{\bar{s},B}(x,z) \sqrt{k(z)}\, u(z)\, dz + \int_{B(c,R)} \nabla_x G^{\bar{s},B}(x,y)\, f^0(y, \sqrt{k}\, u)\, dy$$
$$\frac{u(x)}{2k(x)} \nabla_x k(x). \tag{5.14}$$

Similar to Sec. 3.2.5, the value of $\nabla_x u(x)$ only needs to be estimated for the first ball in any walk—the solution values u, on ∂B and inside B, that the gradient estimate depends on can be computed recursively using the delta tracking or next flight estimator. The parameters $\bar{s}$, p^B, $p^{\partial B}$, P^B and $P^{\partial B}$ remain unchanged from Sec. 5.2 with either estimator.

5.3.2 Reflecting Boundary Conditions

It is straightforward to show how reflecting boundary conditions change under a Girsanov transformation. For instance, assume the normal derivative $\partial u / \partial n = h$ is prescribed in Eq. 5.1 on a Neumann boundary ∂W_N. Moreover, from Eq. 5.3, we know that the function $U(x) = \sqrt{k(x)}\, u(x)$ solves a variable screened Poisson equation. Computing its normal derivative yields

$$\frac{\partial U(x)}{\partial n_x} + m^0(x) U(x) = h^0(x), \tag{5.15}$$

$$\text{where} \quad m^0(x) := \frac{1}{2} \frac{\partial \ln(k(x))}{\partial n_x} \quad \text{and} \quad h^0(x) := \sqrt{k(x)}\, h(x).$$

A delta tracking or next flight variant of the walk on stars estimator from Ch. 4 could in principle deal with the Robin boundary conditions in Eq. 5.15, and simultaneously resolve spatial variability in the PDE coefficients (repeating the derivation in Sec. 5.1 gives an integral expression similar to Eq. 5.9). However, the Robin coefficient m^0 in this scenario is not a strictly positive function, which is a key assumption made with the WoSt estimator in Sec. 4.3. We leave a general treatment of variable-coefficient PDEs with reflecting boundary conditions to future work.

Chapter 6

Solver Implementation & Tuning

Similar to a scene in a renderer, a PDE is encoded by a description of the domain boundary ∂W, boundary conditions g, h, ℓ (Eq. 2.1), source term f, and PDE coefficients $k, \overset{\#}{w}, s$ (Sec. 2.1.1 & App. E). We implement inputs as arbitrary callback routines that return a value for any query point $x \in W$. Unlike conventional solvers such as FEM and finite differences, problem inputs need not be discretized or approximated in a finite-dimensional basis. The gradient and Laplacian of the coefficients $k(x)$ and $\overset{\#}{w}(x)$ (Eq. 5.3 & E.8) can be evaluated via any standard technique (e.g, automatic differentiation), while the bounding parameter $\bar{s} := \max(s^0(x)) - \min(s^0(x))$ is computed as in volume rendering [188], e.g, by regular, random or progressive sampling [169].

In this chapter, we describe how to efficiently implement the geometric queries on ∂W needed by the walk on spheres and walk on stars algorithms (Sec. 6.1 & 6.2). In Sec. 6.3, we detail the impact of the #-shell parameter on the estimated PDE solution with Dirichlet, Neumann or Robin conditions, and discuss convergence with increasing number of walks. We also provide a reference implementation of these PDE estimators in an open-source solver called Zombie [222] (in homage to "random walks").

Figure 6.1: Bézier curves with two-sided boundary conditions and a source term (left) define a diffusion curve image (right). Monte Carlo allows us to zoom in on a region of interest without computing a global solution; there is no loss of fidelity, as curves are not discretized.

All estimators can evaluate the solution and its gradient at arbitrary points in the domain without a global solve. Since values are estimated independently at each point, the implementation is embarrassingly parallel and output-sensitive, i.e., the solve can be performed at any resolution locally (Fig. 1.9, 6.1, 8.4 & 8.6), rather than always on the entirety of a background grid. Moreover, as in rendering, we can progressively increase the number of walks per point to generate a "preview" that can be subsequently refined (Fig. 1.2, 1.9 & 4.2). This approach provides a fast iteration cycle (Fig. 6.2 shows an example where the geometry and boundary conditions are edited interactively), and is especially valuable when working with massive models (Sec. 8.1).

6.1 Closest Point Queries For Walk On Spheres

To find an empty ball $B(x, R)$ around a given point $x \in W$ for the WoS algorithms in Sec. 3.2 and Ch. 5, we need only determine the distance R to the closest point $\bar{x} \in \partial W$, or a conservative underestimate of this distance. As detailed here, such distances are easily computed for a wide

ALGORITHM 6: Closest Pt(T Bvh(¶W_D), x, $\bar{x}$ null, d ¥, d_T^{min} 0)

Input: Bounding volume hierarchy T, query point x 2 R^3, closest point $\bar{x}$ to be determined, current estimate d for distance, and minimum distance d_T^{min} to T's axis aligned bounding box from x (d_T^{min} = 0 when x 2 T.aabb).

Output: Distance to ¶W_D, and closest point on ¶W_D.

```
if d_T^min > d then return d, x̄                          . Ignore current BVH node if it is further than d
if T.isLeaf then
    for t in T.triangles do
        d_t, x̄_t  Closest Pt(t, x)                        . Compute closest pt x̄_t on triangle t from x [62, Sec.5.1.5]
        if d_t < d then d, x̄  d_t, x̄_t                    . Update d and x̄ if x̄_t is closer than d
else
    visitLeft, d_left^min, d_left^max  Intersect(T.left, x, d)   . Check if left aabb intersects ball B(x, d)
    if visitLeft then d  min(d, d_left^max)                 . d cannot be greater than max distance to left aabb
    visitRight, d_right^min, d_right^max  Intersect(T.right, x, d)   . Check if right aabb intersects ball B(x, d)
    if visitRight then d  min(d, d_right^max)               . d cannot be greater than max distance to right aabb
    if visitLeft and visitRight then
        if d_left^min < d_right^min then                    . Visit closer node rst
            d, x̄  Closest Pt(T.left, x, x̄, d, d_left^min)
            d, x̄  Closest Pt(T.right, x, x̄, d, d_right^min)
        else
            d, x̄  Closest Pt(T.right, x, x̄, d, d_right^min)
            d, x̄  Closest Pt(T.left, x, x̄, d, d_left^min)
    else if visitLeft then d, x̄  Closest Pt(T.left, x, x̄, d, d_left^min)   . Visit left node
    else if visitRight then d, x̄  Closest Pt(T.right, x, x̄, d, d_right^min)   . Visit right node
return d, x̄
```

variety of shapes; multiple shapes can be combined by taking the minimum over all per-shape distances, or more generally, by applying Boolean operations (discussed below).

Closest point queries (CPQs) can be accelerated via a spatial hierarchy [203, Ch. 4]. Relative to ray tracing [197, 259], there has been little work on high-performance CPQs [138, 232, 273, 287], though recent GPUs provide opportunities for massive acceleration [260]. We employ an axis-aligned boundary value hierarchy built using the surface area heuristic (SAH) [261], via the FCPW library [220]—unlike ray tracing, oriented bounding volumes [62, 143] can provide more effective culling, and hence further speedups. Alg. 6 provides pseudocode for closest point traversal with a binary BVH containing triangle primitives. The queries described in Sec. 6.2 for WoSt will follow the same basic traversal strategy (Alg. 9), but will use slightly modi ed criteria to determine whether a node in the hierarchy should be visited (Alg. 10).

Polygon Soup. Real-world geometry is often given as a list of polygons without explicit connectivity, and which satis es no special conditions (manifold, orientable, etc.). We can solve PDEs with Dirichlet conditions directly on such “polygon soups” by taking the closest point among all polygons. Rather than attempt to x cracks, holes, and self-intersections [8, 233], we simply solve an exterior problem (Sec. 3.2.4), as shown in Fig. 1.7 (right). Note that in contrast to generalized winding numbers [114], the input need not meet any special conditions on orientation.

Parametric Curves & Surfaces. We use the method of Chen et al. [37] to compute closest points on 2D cubic Bézier curves. Such a representation is attractive for illustration tools (e.g., Illustrator or Inkscape), as it avoids mesh generation and quantization error (consider Fig. 6.1 & 8.7). Closest points can also be computed directly for NURBS and subdivision surfaces [58, 158, 249].

Implicit Surfaces. Many shapes are concisely described by the zero level set of a function $f : \mathbb{R}^N \to \mathbb{R}$. When f is a signed distance function, the size of the largest ball around x is simply $|f(x)|$. More generally, conservative estimates of the distance to an implicit surface can be obtained by bounding the gradient $|\nabla f|$, which in turn gives a Lipschitz constant for f [97, Theorem 1]. A rich variety of shapes can be described this way [97, Table 1]; Fig. 1.7 (left) shows a smooth blend between two tori.

Booleans. Boolean operations are used to concisely encode complex models, via constructive solid geometry (CSG) [209]. Tremendous effort has been put into developing robust mesh booleans [119, 200, 281], but they generally remain expensive and error prone, cannot be mixed with other geometric representations, and still require meshing of the domain interior. In contrast, ray tracing can evaluate booleans via simple arithmetic on intersection distances [212]. We can likewise combine closest point distances to solve PDEs with Dirichlet boundary conditions directly on boolean arrangements (Fig. 1.7 & 6.2). We refer to Hart [97, Table 1] for operations on distances needed for both hard booleans and soft "blends".

Figure 6.2: Our Monte Carlo approach operates directly on the original scene representation (here, signed distance fields composed via CSG operations), and provides instant feedback after updates to the scene geometry and boundary conditions (edit 1) or PDE coefficients (edit 2).

6.2 Accelerated Geometric Queries For Walk On Stars

In general, WoSt works with any boundary representation that supports the following queries:

1. closest point queries on ∂W_D,
2. closest silhouette point queries on ∂W_N,
3. star region radius queries on ∂W_R,
4. ray intersection queries against ∂W_N and ∂W_R, and
5. point sampling queries on ∂W_N and ∂W_R.

Since none of these queries require the boundary to have a well-defined inside and outside, ∂W need not be watertight, and can have cracks, holes, or self-intersections—see in particular App. B for a discussion of open domains and double-sided boundary conditions.

In principle, these queries could be evaluated for, say, spline patches or implicit surfaces (as in Sec. 6.1); we focus exclusively on triangle meshes here. In particular, ray intersections are standard in computer graphics, and can be accelerated using a BVH (just like CPQs). Sec. 6.2.1 describes the closest silhouette point query for Neumann boundaries. The star region radius

Figure 6.3: Left: An optimized procedure for nding silhouettes should avoid visiting nely-tessellated geometry that is entirely front- or back-facing relative to the query point. Right: A SNCH tests for a pair of mutually orthogonal directions in a view cone and a node's normal cone to determine whether the node contains a silhouette edge on the Neumann boundary. The geometry inside the node can be skipped if no such pair of directions is found.

query for Robin boundaries, detailed in Sec. 6.2.2, functions as a CSPQ when the Robin coef cient $m = 0$, and as a CPQ when $m = \infty$. For intermediate values of m, this query has additional aspects unique to Robin conditions, which we highlight in green in Alg. 8 & 10. The point sampling query (Alg. 11) is designed to sample known Neumann data h and Robin data $\grave{}$ on re ecting boundaries, and is discussed in Sec. 6.2.4.

All queries use a spatial hierarchy to visit only a small fraction of the triangles on the boundary of the input domain; our basic approach is to build a spatialized normal cone hierarchy (SNCH) [119] by adding normal information to the BVH already used by WoS. In particular, we use a standard BVH to perform CPQs on the Dirichlet boundary, and a separate SNCH for queries on the Neumann and Robin boundaries, using normal information only for queries 2 & 3. In practice, all queries needed to implement WoSt on triangle meshes are supported by the open-source FCPW library [220]; see also App. F for pseudocode.

6.2.1 Closest Silhouette Point Query For Neumann Boundaries

The silhouette of a triangle mesh, relative to a given direction v, occurs along a set of edges e that satisfy a local silhouette condition. In particular, e is a silhouette edge if for each distinct pair of triangles containing e,

$$(v \quad n_1) \quad (v \quad n_2) \quad 0, \tag{6.1}$$

where n_1, n_2 are consistently oriented normals (see inset). Note in particular that every boundary edge is a silhouette edge. A na¨ve strategy for nding the closest silhouette edge is to use a BVH to locate the closest point to x on all edges, skipping edges not contained in the silhouette. However, this strategy is highly inef cient when BVH nodes contain large, nely-tessellated regions that are all front- or back-facing (Fig. 6.3, left): here each edge is examined (and rejected) exhaustively, whereas ideally the whole node should simply be culled.

Spatialized Normal Cone Hierarchy

To improve scaling, we hence augment our BVH with information about the orientation of the geometry inside each node. In particular, we adopt the spatialized normal cone hierarchy of Johnson and Cohen [119]. Each node of a SNHC stores not only an axis-aligned bounding box (AABB), but also a normal cone The cone axis is the average normal of all triangles in the node,

and the cone half angle θ is the maximum angle between the axis and any triangle normal (Fig. 6.3, right). Normal cones can be assembled during BVH construction. We use the surface area heuristic; performance could possibly be further improved via the surface area orientation heuristic (SAOH) of Conty Estevez and Kulla [39, Sec. 4.4], which clusters primitives according to both proximity and alignment.

Closest Silhouette Point Traversal

To perform a silhouette query on ∂W_N, we traverse the SNCH in depth-first order (Alg. 9). For each node in this traversal we build a view cone rooted at x. The cone's axis points toward the center of the node, and its half-angle tightly bounds the AABB (Fig. 6.3, right); we then check if the view cone and the node's normal cone contain a pair of mutually orthogonal directions. If this test fails, all triangles in the node must be front- or back-facing relative to the query point, and the node can be skipped. In the context of WoSt, an upper bound on the size of a star-shaped region $\mathrm{St}(x, R)$ is given by the distance $d_{\mathrm{Dirichlet}}$ from x to the Dirichlet boundary (Sec. 4.2.2). To further improve query efficiency we can hence restrict the search to the radius $R^{\max} = d_{\mathrm{Dirichlet}}$.

6.2.2 Star Region Radius Query For Robin Boundaries

The naïve approach for computing the radius of a star-shaped region on ∂W_R is to first perform a CSPQ relative to a query point x, and then to loop over all triangles within the radius returned by the CSPQ, estimate the upper-bound on the radius for each triangle t (Eq. 4.11), and take the minimum of these bounds. Before we describe how to accelerate this computation using an SNCH, we observe that the radius bound does not have to be approximated numerically for any t. We can instead compute a tight bound using the maximum coefficient value $\bar{m}^{\max} := \max(\bar{m}(z))$[1] over all points $z \in t$, and a distance h from x to the plane t lies on. In particular, letting $r = h/\cos\theta$ in Eq. 4.11, we have:

$$R \leq \frac{\bar{m}^{\max} h^2}{\bar{m}^{\max} h \cos\theta - \cos^3\theta} \quad \text{when } \cos\theta \geq \sqrt{\bar{m}^{\max} h}. \tag{6.2}$$

We now minimize this equation by taking its derivative with respect to $\cos\theta$, and setting it to zero. This gives an analytical expression $\sqrt{\bar{m}^{\max} h/3}$ for the cosine. We clamp this expression between the minimum and maximum cosine values achieved at the closest and farthest points on t (respectively), and plug it back into Eq. 6.2 to compute the radius bound for t. Alg. 8 provides pseudocode.

6.2.3 Accelerating Star Region Queries

Not every triangle in a mesh needs to be visited to compute the radius of a star-shaped region. In fact, as we search for the minimum upper-bound on the radius over all triangles, we can skip over large parts of ∂W_R where the geometry is entirely front- or back-facing relative to the query point x (akin to a CSPQ). The only alteration we make to the SNCH from Sec. 6.2.1 is to also store minimum and maximum Robin coefficient values over all triangles in a node.

[1] Using $\bar{m}^{\max}$ to compute the radius bound for a triangle does not alter the original problem description—the WoSt estimator will still treat $\bar{m}(z)$ as spatially-varying over the triangle when computing rectance in Alg. 3, line 1.

Query Traversal. Similar to Eq. 6.2, we compute conservative radius bounds for the nodes we visit during traversal, using the spatial, angular, and coefficient information available in a SNCH node (Alg. 10, lines 7-10). We also build a view cone rooted at x to compute the bounds, via Eq. 4.11. We then decide whether a node with only front or back-facing triangles can be skipped, by checking if our minimum bound for the node is larger than the current estimate of the radius for the star-shaped region (Alg. 9, line 1). We also use our maximum bound for the node to shrink the radius estimate (Alg. 9, lines 7 & 9). If instead the node contains a geometric silhouette (Alg. 10, line 4), we must traverse the node just as with a CSPQ, since Eq. 4.11 only applies to star-shaped regions which can be no larger in size than the distance returned by a CSPQ.

6.2.4 Point Sampling Query

Recall that for problems with nonzero Neumann and Robin conditions, we sample points on ∂W_N and ∂W_R respectively (Sec. 4.2.3). To increase the likelihood that these points lie on ∂St_N and ∂St_R, we adopt a hierarchical importance sampling strategy used in rendering to accelerate next-event estimation [39]. In particular, during each step of a BVH traversal, we select only a single random child whose AABB intersects $St(x, R)$. To give preference to nodes closer to the query point x, we sample according to the free-space Green's function G^{R^N} (Alg. 12, lines 11-14), rather than the Green's function for a ball (which becomes negative outside $St(x, R)$). Once we reach a leaf node, we uniformly sample a point from the leaf triangles with respect to surface area (Alg. 11, line 3). This point is not guaranteed to lie on ∂St_N or ∂St_R, but is much more likely to do so compared to uniformly sampling all of ∂W_N or ∂W_R (respectively). Conty Estevez and Kulla [39, Sec. 5.4] describe further improvements to this traversal strategy.

6.3 Epsilon Parameter & Convergence Of Estimators

Our solvers use a single parameter to control the thickness of the epsilon shell $\partial W^{\#}$, irrespective of the type of boundary condition prescribed on ∂W. This parameter trades bias in the solution estimate with the number of steps in a random walk, and in general requires little-to-no hand-tuning, as bias drops predictably with decreasing values of $\#$.

Dirichlet Boundary. Early stopping extends boundary values into an epsilon-neighborhood $\partial W_D^{\#}$, and the solution in turn exhibits a small bias toward these values. Dirichlet boundary conditions are hence still enforced with the given data, but the location of enforcement may be off by a tiny distance $\#$. Though in principle bias can be completely eliminated via a Green's function first passage approach [81, 159], a pragmatic solution is to simply use a small value of $\#$.

In practice, an accurate solution is obtained for fairly large $\#$ values, even in the presence of tiny features (Fig. 6.4). Importantly, arbitrarily small geometric features will always appear in the solution, since there will always be evaluation points x in their Voronoi region, independent of $\#$. Moreover, such features will still have a global effect on the solution, since a random walk has a nonzero probability of reaching any boundary component of finite size. The only potential problem is if spacing between features is smaller than $\#$, which is easily avoided by (universally or adaptively) using a small $\#$. Note that shrinking $\#$ by a few orders of magnitude does not significantly increase cost due to the exponential shrinking of balls [22]. In contrast, mesh-based solvers can eliminate small features entirely, and their time/memory cost often blows up dramatically with smaller tolerances $\#$ (Fig. 1.3, 8.1, 8.4 & 8.10).

Figure 6.4: Tiny features are preserved for any value of $\#$. For very large values of $\#$, WoS algorithms jump to the closest point, producing a Voronoi-like solution (left). Decreasing $\#$ quickly eliminates any bias. Since small $\#$ values do not signi cantly increase the average number of steps per walk, it is generally unnecessary to hand-tune this parameter. (Note that if the box above represents a domain 1 meter in width, then 10^{-4} is about the width of a human hair; small bacteria are on the order of 10^{-6}.)

Neumann And Robin Boundaries. Enforcing a minimum radius on the size of a star-shaped region (Sec. 4.2.2) also incurs bias near concave parts of $\P W_N$ and $\P W_R$. This bias manifests as a global darkening of the estimated solution as $\#$ is increased. Fig. 6.5 examines the effect of this parameter on a Neumann problem—compared to Dirichlet conditions, here the performance-bias tradeoff is more sensitive to values of $\#$. However as before, the relative rate at which bias grows with increasing $\#$ values is outweighed by the performance improvement from walks taking larger steps on $\P W_N$, and terminating faster on $\P W_D$. Adaptively picking $\#$ based on local boundary curvature should yield better performance and lower bias.

Figure 6.5: WoSt uses an $\#$-shell to also ensure that random walks make progress near concave parts of a re ecting boundary. Walks generally converge faster with larger values of $\#$, with run-time improvements outweighing the relative increase in bias.

Fig. 6.6 examines the impact of $\#$ on a Robin problem with both more absorbing ($m > 1$) and more re ecting ($m < 1$) boundary conditions—in the limit, we recover the affect of $\#$ on pure Dirichlet and Neumann problems respectively. In all other experiments, we scale models to t in a unit sphere, and use $\# = 0.001$ with all boundary conditions.

Convergence. All PDE estimators from Ch. 3–5 exhibit the expected $O(1/\sqrt{N})$ Monte Carlo rate of convergence with respect to the number of walks N (Sec. 3.1), suggesting that any bias from the sole $\#$-shell parameter has little impact on overall accuracy; see Fig. 5.4, 6.6, 8.2, 8.14, 8.16 & 8.17. In general, variance tends to be larger (but still predictable) with recursive estimation of higher-dimensional integrals, i.e., it is larger in regions where walks are longer—examples include Neumann dominated problems (Fig. 8.17, bottom center), and PDEs with high spatial variability in their coef cients (Fig. 7.20). We describe variance reduction strategies for these estimators in the next chapter, which do not increase the rate of convergence, but they do lower the magnitude of the error in the estimated results (e.g, Fig. 7.4, 7.17 & 7.20).

Figure 6.6: Top two rows: For more reflecting Robin boundaries with smaller coefficients, bias manifests as a global darkening in the solution estimate for large t. Bottom row: For more absorbing Robin boundaries with larger coefficients, a large t-shell extends prescribed boundary values further into the domain interior, similar to the bias observed with pure Dirichlet problems.

Chapter 7

Variance Reduction

Like any other Monte Carlo method, we can reduce the noise in our PDE estimators by simply taking more samples, i.e., by performing more random walks from an evaluation point. However, in many application scenarios, the solution must be estimated within a fixed time constraint, or using a given sample budget. Therefore, it is often not enough to just design Monte Carlo estimators—we must also look for ways to make them more efficient (Eq. 3.5).

In this chapter, we develop several variance reduction strategies for the estimators developed in Ch. 3–5, by leveraging the special structure of integral equations corresponding to elliptic PDEs. Some of these strategies are inspired by analogous methods in Monte Carlo rendering as the integrals share similarities (e.g, Sec. 7.1, 7.4, 7.5 & 7.6), while others are specifically designed for elliptic equations (e.g, Sec 7.2, 7.3 & 7.7). Most strategies are complementary to each other (and can hence be used in conjunction), as they target different facets of our estimators. Though we achieve noticeable improvement in estimation quality here, there remains significant scope for further variance reduction—as will be discussed in Ch. 9.

7.1 Importance Sampling Of Source Terms

Recall that the integral formula for a Poisson equation involves a term $\int_{B(x,R)} G^B(x,y)\, f(y)\, dy$, where G^B is the harmonic Green's function on $B(x,R)$. One way to importance sample this term is to simply draw y from the distribution $p^B := G^B / \int_{B(x,R)} G(x,y)\, dy$, as previously discussed in Sec. 3.2.1. Such samples can be generated via standard techniques, e.g, rejection sampling (App. A. 1.2 & A. 2.2); in 3D we use Ulrich's polar method [54, Section 9.4]. In principle, one could extend this strategy to PDEs where the Green's function can only be tabulated numerically, akin to importance sampling of measured BRDFs [145].

Figure 7.1: Uniform (left) vs importance sampling (right) of the Green's function.

We can also sample the source term f. An important case is a point source $f := c\delta_z$, where $c \in \mathbb{R}$ is a constant, and δ_z is the Dirac delta centered at a point $z \in W$. If a single δ_z is inside the current ball $B(x,R)$, we can use an importance density $p^B = \delta_z$, yielding the estimator

$$\frac{G^B(x,y)\, f(y)}{p^B(y)} = \frac{G^B(x,y)\, c\delta_z(y)}{\delta_z(y)} = cG^B(x,z), \tag{7.1}$$

i.e., just use a single sample at $y = z$. Similarly, consider a curve source $f := c\delta_g$, where δ_g is the (1-Hausdorff) measure of a curve $g \subset W$, and c is a function along g. When g intersects B, we can sample points $y_1, \ldots, y_M$ uniformly from $g_B := g \setminus B$, and use the estimator

$$|g_B| \frac{1}{M} \sum_{i=1}^{M} c(y^i) G(x, y^i). \tag{7.2}$$

Alternatively, one can just uniformly sample the whole curve g, and drop the contribution of

Figure 7.3: Here we use multiple importance sampling [252] of the Green's function G^B and source term f to robustly sample screened Poisson equations for different coefficient values with area sources of varying size.

points outside B. This strategy is easily generalized to any m-dimensional subset; Fig. 7.2 shows an example.

The importance sampling strategies described above, with G^B and f as the sampling densities, can also be combined using multiple importance sampling [252], as in Fig. 7.3. We note that Monte Carlo is often better suited than quadrature methods for integrands with singularities—importance sampling can be applied to handle such integrands effectively, even in situations where there is no analytic transformation to remove the singularity.

Figure 7.2: Source terms can be importance sampled by randomly picking points on the source where it is non-zero, restricted to the current ball in a random walk. Without importance sampling, only the area source f_A would appear on the right; the point source f_z and curve source f_g would never get sampled.

7.2 Control Variates

Suppose we let $\tilde{f}$ be a low-order Taylor approximation of f in Eq. 3.7 around some point x_0. Then the function $f - \tilde{f}$ will look "flat" in a small neighborhood of x_0, and estimating it will yield low variance. This control variate is useful for WoS estimators, which seek to integrate a function over a typically small sphere or ball. It also applies to WoSt if we use an empty ball, in place of a star-shaped region, for just the first step of a random walk. Though we do not know the terms of the Taylor series a priori, we can use running derivative estimates to get an increasingly good guess. In fact, when both the solution and gradient strategies decribed below are used in conjunction, they reinforce each other: the variance of the solution estimator is reduced by the gradient estimate, and vice versa. In practice, we use these two strategies for all PDEs.

Control Variate For Solution

For a PDE with solution u, let $\hat{\nabla}^d u_k(x_0)$ be the running estimate of the gradient for the first k walks at x_0 (computed as in Sec. 3.2.5). Since the linear term $\nabla u(x_0) \cdot (x - x_0)$ in the Taylor series for $u(x_0)$ integrates to zero over any ball around x_0 (i.e., $c = 0$ in Eq. 3.7), we can replace an

Figure 7.4: Control variates provide modest variance reduction for the solution, but become more important for higher derivatives of the solution. Here we plot the Hessian's principal values and axes as ellipses; notice that variance is higher near the boundary, and axes are harder to estimate in regions where the Hessian has small magnitude.

existing WoS estimator $\hat{u}$ with

$$\frac{1}{N}\sum_{i=1}^{N} \hat{u}(x_1^i) \quad \hat{d}\, u_{i\ 1}(x_0) \quad (x_1^i \quad x_0), \quad x_1^i \quad U\,(\P B(x_0, R)), \tag{7.3}$$

where x_1^i denotes the rst step in the ith walk. Note also that the estimate $\hat{d}\, u_{i\ 1}$ is statistically independent of the ith walk. In practice even this simple strategy can help reduce variance—see for example Fig. 7.4.

Control Variate For Gradient

Variance reduction for derivatives is especially important, since differentiation ampli es high frequencies. Our control variate strategy for the gradient is complementary to the one for the solution: let $\hat{u}_k(x_0)$ be the running estimate of the solution $u(x_0)$ for the rst k walks. Then we can replace the gradient estimate for the boundary term (Eq. 3.18) with

$$\frac{d}{R}\frac{1}{N}\sum_{i=1}^{N} (\hat{u}(x_1^i) \quad \bar{u}_{i\ 1}(x_0)) \quad n_{x_1^i}. \tag{7.4}$$

Since by symmetry the normal $n_{x_1^i}$ integrates to zero over a sphere, the expected value of the estimator is unchanged (i.e., c = 0), but the variance is typically lower as the control variate approaches the true value of $u(x_0)$. Likewise, if $f_k := \frac{1}{k}\sum_{i=1}^{k} f(y^i)$ is the running average of source values sampled from the initial ball, then we can subtract this value from $f(y)$ in our estimator for the initial source term in Eq. 3.14.

This strategy parallels methods used in reinforcement learning [245, Section 13.4], and is related to techniques used on discrete grids [183]. We refer to Rioux-Lavoie et al. [211, Sec. 4] for an antithetic variate strategy (Sec. 3.1.1) for the gradient estimator, which can be used alongside our control variate to reduce variance even further. Both strategies are particularly effective when estimating the gradient in the vicinity of the domain boundary, where the ball size is smaller.

Control Variate For Hessian

Control variates can also be applied to higher-order derivatives. For example, in the ith term of our Hessian estimator for the boundary integral (Eq. 3.23), we can replace $u(x_1^i)$ with

$$u(x_1^i) \quad u_{i\ 1}(x_0) \quad \nabla u_{i\ 1}(x_0) \quad (x_1^i \quad x_0). \tag{7.5}$$

Alanko and Avellaneda [3] discuss a similar approach for grids. Note that both running sums have already been computed for the source and gradient control variates—these could now be further improved via the Hessian estimate. Fig. 7.4 shows the effect on variance, which is about 7x lower than the baseline estimator from Eq. 3.23.

7.3 Tikhonov Regularization

The solution to a Poisson equation with pure Neumann conditions is determined only up to an additive constant. When we solve such a PDE with WoSt, we observe that high frequency details in the PDE solution are often resolved by the first few steps of a random walk, while the contribution from later steps is closer to constant; see Fig. 3.6. As mentioned in Sec. 4.2, we use Tikhonov regularization (Sec. 3.2.6) to more effectively handle such problems—Fig. 7.5 shows this approach provides estimates without much noise or bias even with substantial regularization, while ensuring walk length is not unbounded. In general, the number of steps needed to resolve the solution is problem-dependent, and more steps are typically needed when the solution has low-frequency global features.

Figure 7.5: Top: A small Tikhonov parameter τ yields long walks and high variance, while larger τ values produce shorter walks with less noise but more bias. Bottom: Since the solution is often well-resolved by short walks, we apply regularization only to walks longer than a given length—yielding lesser noise and bias.

7.4 Adaptive Sampling And Denoising

Elliptic PDEs have highly regular solutions away from the source and boundary. Conventional methods exploit this behavior by interpolating over a mesh; we can likewise interpolate over scattered samples to dramatically reduce cost. For instance, in Fig. 7.7 (center), we use simple Poisson disk sampling [30] and moving least squares (MLS) interpolation [181]. To avoid "bleeding" artifacts, we shoot rays to exclude neighbors not visible from sample points. There are plenty of opportunities for improvement and acceleration (e.g, via fast multipole methods [244]); note that all interpolation schemes introduce bias (including standard nodal interpolation in FEM).

We can also use adaptive sampling to concentrate effort on interesting regions. For instance, Fig. 7.7 (right) applies a simple scheme in the spirit of irradiance gradients [265]: we first estimate the solution and gradient at a set of seed points. Then, for any candidate sample x, we evaluate a 1st-order Taylor approximation $u(y_i) + \nabla u(y_i), x \quad y_i i$ for the k nearest neighbors $y_1, \ldots, y_k$ of x. If the sample variance (Eq. 3.4) divided by the mean is above a threshold $a > 0$, we estimate the value and gradient at x and add it to the set. More sophisticated strategies from rendering provide ample inspiration for future work [288].

Figure 7.7: Just as mesh-based methods interpolate solution values at a few sparse points, we can rapidly visualize solutions to PDEs via scattered data interpolation. Here we solve a Laplace problem using either uniform or adaptive sampling and simple MLS interpolation. Adaptive sampling better resolves high-frequency boundary conditions.

Techniques for denoising rendered images [123, 137, 189, 190, 213, 226] also translate well to the PDE setting. For instance, in Fig. 7.6, we apply Intel's deep learning-based Open Image Denoise algorithm [112], using the boundary value at the closest point in place of the albedo map (no other results in this work use denoising). Training such a network on PDE data, rather than rendered images, should further reduce bias and improve performance.

Figure 7.6: Even for a small number of walks per pixel (left), techniques for denoising renders (center) closely match the reference solution (right), further increasing efficiency—especially for PDEs with smooth solutions.

7.5 Boundary Value Caching

Grid-free Monte Carlo methods such as walk on spheres independently estimate the solution at every point, and hence do not take advantage of the high spatial regularity of solutions to elliptic problems. Here we describe a fast caching strategy [166] for the boundary integral

$$a(x)\,u(x) = \underbrace{\int_{\partial W} P^{\mathbb{R}^d}(x,z)\,u(z) + G^{\mathbb{R}^d}(x,z)\,\frac{\partial u(z)}{\partial n_z}\,dz}_{=:\,u_{\partial W}(x)} + \underbrace{\int_{W} G^{\mathbb{R}^d}(x,y)\,f(y)\,dy}_{=:\,u_{W}(x)} \tag{7.6}$$

of a (screened) Poisson equation in a domain W, where $P^{\mathbb{R}^d}$ and $G^{\mathbb{R}^d}$ are free space functions, rather than functions for the ball. To make use of this BIE, one must somehow determine the unknown boundary data: Dirichlet values u on the Neumann boundary ∂W_N, Neumann values $\partial u/\partial n$ on the Dirichlet boundary ∂W_D, and both, or one of, u and $\partial u/\partial n$ on the Robin boundary ∂W_R. Schemes such as the boundary element method use a finite-dimensional space of functions on the boundary (e.g, basis functions associated with mesh nodes), and solve a dense, globally-coupled linear system for the best approximation to the true solution.

BVC takes a completely different approach, and instead uses WoSt to compute the unknown boundary values. It hence avoids global solves, boundary remeshing, and approximation of the function space; unlike BEM, it also handles the source term f. Moreover, as random walks can be expensive (especially in problems with predominantly Neumann boundaries), we cache these boundary values at a collection of random sample points along ∂W. We then use a Monte Carlo estimate of Eq. 7.6 for cheap, output-sensitive evaluation of the solution (or its gradient) at any interior point $x \in W$, without taking any further random walks (Sec. 7.5.1).

Figure 7.8: We obtain far smoother results with BVC compared to directly using pointwise estimators like WoSt at equal time. BVC uses the same boundary samples to determine the PDE solution across the domain.

Overall, this scheme is easy to parallelize, and can be computed progressively (e.g, for interactive preview). It can handle imperfect geometry (e.g, with self-intersections) and detailed boundary/source terms without repairing or resampling the boundary representation (Fig. 8.15). We can also focus computation on a region of interest by caching points only on the boundary of a small subdomain $L \subset W$ (Sec. 7.5.2)—unlike BEM which must perform a global solve over the entire boundary ∂W.

Figure 7.9: BVC gradients have considerably less noise compared to pointwise estimates, as they use known values of $\partial u / \partial n$ on the Neumann boundary. In contrast, WoSt gradients become noisier away from the Dirichlet boundary, as estimation requires longer random walks.

In practice, we obtain far smoother results across the domain compared to directly using pointwise estimators like WoS or WoSt (Fig. 7.8, 7.9 & 7.10). This behavior can be attributed to correlations in the solution estimates at interior evaluation points that use the same boundary and source samples. On the flip side, error is now more global akin to traditional PDE solvers such as FEM and BEM (Fig. 7.16 & 7.17). Unlike pointwise estimators, we also observe boundary artifacts (Fig. 7.15), as samples are no longer generated in proportion to the singular functions P^{R^d} and G^{R^d}. We show how to mitigate such artifacts in Sec. 7.5.3.

Sample Reuse In Rendering. Popular sample reuse schemes in rendering such as virtual point lights [46, 132], photon mapping [91, 92, 115] and ReSTIR [23, 193] share samples across pixels to amortize the cost of long ray traced paths, and inject global information into per-pixel radiance estimates. As a result, they are often more efficient at rendering scenes with complex geometry and lighting than brute-force path tracers.

BVC shares similarities with VPLs and photon mapping, in that samples generated and deposited on the scene (for us ∂W) determine the radiance (i.e., the solution u) over the image plane (which for us is either the entire domain W or a subset thereof). Unlike photon mapping however, BVC does not require an additional data structure like a kd-tree to store samples; instead we opt for a progressive formulation that discards boundary and source samples after splatting solution and gradient estimates in W. Similar to VPL methods, Monte Carlo noise is visually

Figure 7.10: Boundary value caching dramatically reduces the total number of random walks needed to solve PDEs relative to pointwise Monte Carlo estimators. Here we show streamlines of a potential ow in a simulated wind tunnel, computed directly from a low-quality surface mesh originally intended for visualization rather than simulation.

suppressed [46, Fig. 1] from a combination of introducing correlations between estimates in W, and the smooth decay of the functions P^{R^d} and G^{R^d} away from the boundary (Fig. 2.3). VPLs are also prone to singularities, as sharing samples requires sacri cing perfect importance sampling [46, Sec. 5 & Fig. 9]; our artifact correction schemes take inspiration from similar techniques for VPLs [135]. Unlike VPL methods, we do not require testing for occlusion between deposited samples and evaluation points as the BIE contains no visibility term.

Techniques based on lightcuts [152, 264, 275] render scenes containing thousands of VPLs in real time. These methods cluster VPLs spatially in a tree, and then probabilistically select a subset of the VPLs that make the largest contribution at a given point. Akin to Fast Multipole and Barnes-Hut schemes [88, 202] for BEM, a lightcuts-based strategy should asymptotically reduce the quadratic complexity of evaluating the BIE with BVC. We leave development of such a strategy to future work.

7.5.1 Monte Carlo Estimation

We estimate the solution $u_{\partial W} + u_W$ in Eq. 7.6 at a set of evaluation points $\text{evalPts} := \{x_k \in W\}_{k=1}^{K}$ in a closed domain $W \subset R^d$ by creating two caches, $\text{boundarySamples} := \{z_i, b(z_i), \tilde{\partial u}/\partial n(z_i)\}_{i=1}^{N}$ and $\text{sourceSamples} = \{y_j, f(y_j)\}_{j=1}^{M}$, where N and M are user-speci ed cache sizes. The points z_i and y_j are sampled on the boundary ∂W and inside the domain W using probability densities $p^{\partial W}$ and p^W respectively. The pointwise estimates $b(z_i)$ and $\tilde{\partial u}/\partial n(z_i)$ are computed using WoSt (Ch. 3, Sec. 3.2.5 & 7.2), while $f(y_j)$ are evaluations of the known source term.

Dirichlet-Neumann conditions. For BVPs with mixed Dirichlet and Neumann conditions, BVC then uses the two caches to form correlated Monte Carlo estimates of Eq. 7.6 at all points in evalPts:

$$d_{\partial W}(x_k) := \frac{1}{N} \sum_{i=1}^{N} \frac{P^{R^d}(x_k, z_i)\, b(z_i) + G^{R^d}(x_k, z_i)\, \tilde{\partial u}/\partial n(z_i)}{p^{\partial W}(z_i)}, \tag{7.7}$$

$$u_W(x_k) := \frac{1}{M} \sum_{j=1}^{M} \frac{G^{R^d}(x_k, y_j)\, f(y_j)}{p^W(y_j)}. \tag{7.8}$$

ALGORITHM 7: A boundary value caching strategy to evaluate a d-dimensional boundary integral equation inside a closed user-defined boundary ∂L

```
struct Boundary Sample
    z, n_z ← null                                   . Sample location and unit outward normal of ∂L
    b.u, b.∂u/∂n ← 0                                . Estimates of solution and normal derivative
struct Evaluation Point
    x ← null                                        . Location for evaluating the BIE
    b.u^sum_∂L, b.u^sum_L ← 0                       . Running sums for solution evaluation
    N, M ← 0                                        . Boundary and source sample count
    function GetSolution()
        return b.u^sum_∂L N + b.u^sum_L M
Input: A set of evalPts, boundary and source cache sizes N and M, nWalks for pointwise estimation, and
    PDFs p^∂L and p^L for sample generation.
Output: An updated solution estimate at each evaluation point.
function UpdateSolution(evalPts, N, M, nWalks, p^∂L, p^L)
    boundarySamples ← Boundary Sample[N]            . Initialize N boundary samples
    parallel for b in boundarySamples do
        b.z, b.n_z ← p^∂L                           . Generate boundary sample from PDF p^∂L
        b.u, b.∂u/∂n ← Walk On Stars(b.z, b.n_z, nWalks)   . Use nWalks to estimate u and ∂u/∂n with WoSt
    for b in boundarySamples do
        z, n_z ← b.z, b.n_z
        parallel for e in evalPts do                . Splat boundary contribution from b to all evalPts
            e.b.u^sum_∂L += P^{R^d}(e.x, z, n_z) b.u + G^{R^d}(e.x, z) b.∂u/∂n ← p^∂L(z)
            e.N += 1
    for j in Range(M) do
        y ← p^L                                     . Generate source sample from PDF p^L
        parallel for e in evalPts do                . Splat source contribution from y to all evalPts
            e.b.u^sum_L += G^{R^d}(e.x, y) f(y) ← p^L(y)
            e.M += 1
```

In App. A, we provide expressions for P^{R^d} and G^{R^d} for the Poisson and screened Poisson equations. Alg. 7 provides pseudocode, and App. B.2 discusses the extension to open domains and double-sided boundary conditions.

Robin conditions. For BVPs with Robin conditions, we need not estimate both u and $\partial u/\partial n$ with WoSt. Instead, substituting $\partial u/\partial n = \grave{} \; m u$ on a Robin boundary ∂W_R in Eq. 7.6 gives

$$a(x)\, u(x) = \int_{\partial W_R} P^{R^d}(x,z) \; m(z) G^{R^d}(x,z) \; u(z)\, dz + \int_{\partial W_R} G^{R^d}(x,z) \; \grave{}(z)\, dz + \int_{W} G^{R^d}(x,y) \; f(y)\, dy, \tag{7.9}$$

for which only the solution u needs to be estimated on ∂W_R. As an alternative, we could also make the substitution $u = (\grave{} \; \partial u/\partial n) / m$ when $m > 0$, and instead solve for unknown values of $\partial u/\partial n$ on ∂W_R. We observe that estimating $\partial u/\partial n$ typically works better when m is large. This is because BVC does not importance sample the term $P \; m G$ in Eq. 7.9 when generating its cache samples on ∂W_R, which means that a large m amplifies noise in estimated values of u on ∂W_R.

Gradient Estimation. We can reuse the same cached boundary and source samples to also form Monte Carlo estimates of the solution gradient at each evaluation point (Fig. 7.9):

$$\frac{\partial u_{\partial W}}{\partial x}(x_k) := \frac{1}{N} \sum_{i=1}^{N} \frac{\frac{\partial P^{R^d}}{\partial x}(x_k, z_i)\, b(z_i) + \frac{\partial G^{R^d}}{\partial x}(x_k, z_i)\, \frac{\partial u}{\partial n}(z_i)}{p^{\partial W}(z_i)}, \tag{7.10}$$

$$\frac{\partial u_{W}}{\partial x}(x_k) := \frac{1}{M} \sum_{j=1}^{M} \frac{\frac{\partial G^{R^d}}{\partial x}(x_k, y_j)\, f(y_j)}{p^{W}(y_j)}. \tag{7.11}$$

We provide expressions for the derivatives of P^{R^d} and G^{R^d} in App. A.

Sampling. We can use a discrete cumulative density function (CDF) table [203, Sec. 13.3] with stratified random numbers to generate boundary samples over elements (e.g., triangles) of a polygonal mesh. Faster sample generation is possible with an alias table [262, 263]. Source samples can likewise be generated uniformly inside W with stratified sampling [203, Sec. 13.8]. We use this sampling setup for all BVC figures in this text except Fig. 7.10, where we find that results improve significantly if the boundary samples in Eq. 7.7 & 7.10 are weighted by the area of their associated Voronoi cell. Similar area weighting strategies have proven effective for surface reconstruction [12, Fig. 5], and yield a consistent Monte Carlo estimator that provides provably better convergence [89].

Progressive Evaluation. BVC is progressive in two ways. First, we can improve estimation quality at a set of evaluation points, by generating new caches and using them to update existing estimates (Alg. 7, UpdateSolution). Second, we can compute solution estimates at new evaluation points by iterating over existing caches.

Bias. Assuming the pointwise estimates $b(z_i)$ and $\partial u/\partial n(z_i)$ are unbiased, the estimators in Eq. 7.7–7.11 are also unbiased via the linearity of expectation. In reality, most pointwise estimators have a small amount of controllable bias from the $\#$-shell (Sec. 6.3). Unlike MLS interpolation and adaptive sampling (Sec. 7.4), BVC does not introduce any additional bias, while improving efficiency noticeably (Fig. 7.8).

7.5.2 Boundary Specification

When the solution needs to be evaluated within a localized region L inside the domain W (Fig. 7.11), we specialize the BIE to this region by generating source samples in L and boundary samples on ∂L (Alg. 7, lines 13 & 21). We use uniform densities $p^{L} = 1/|L|$ and $p^{\partial L} = 1/|\partial L|$ for sample generation, though we could use densities specialized to a specific PDE to reduce variance further. The solution integrates to 0 outside L by construction.

When the solution needs to be evaluated within the entire domain W (i.e., $L = W$), we incorporate the known boundary data $\partial u/\partial n = h$ on ∂W_N directly into our sample estimates,

Figure 7.11: We can focus computation on a region of interest by caching points on the region boundary.