Toward General-Purpose Monte Carlo PDE Solvers for Graphics Applications

by

Ryusuke Sugimoto

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Ioannis Gkioulekas Associate Professor Robotics Institute, School of Computer Science Carnegie Mellon University
Supervisors:	Toshiya Hachisuka Associate Professor David R. Cheriton School of Computer Science University of Waterloo
	Christopher Batty Associate Professor David R. Cheriton School of Computer Science University of Waterloo
Internal Members:	Yuying Li Professor David R. Cheriton School of Computer Science University of Waterloo
	Justin W.L. Wan Professor David R. Cheriton School of Computer Science University of Waterloo
Internal-External Member:	Stanislav Potapenko Associate Professor Department of Civil and Environmental Engineering University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis comprises materials from the following works, where I, Ryusuke Sugimoto, was either the lead author or a joint-lead author:

(a) Rioux-Lavoie, Sugimoto, Özdemir, Shimada, Batty, Nowrouzezahrai, and Hachisuka [2022], "A Monte Carlo Method for Fluid Simulation," SIGGRAPH Asia 2022 / ACM Transactions on Graphics (joint first authors),

(b) Sugimoto, Chen, Jiang, Batty, and Hachisuka [2023], "A Practical Walk-on-Boundary Method for Boundary Value Problems," SIGGRAPH North America 2023 / ACM Transactions on Graphics,

(c) Sugimoto, Batty, and Hachisuka [2024a], "Velocity-Based Monte Carlo Fluids," SIGGRAPH North America 2024 Conference Papers,

(d) Sugimoto, King, Hachisuka, and Batty [2024b], "Projected Walk on Spheres: A Monte Carlo Closest Point Method for Surface PDEs," SIGGRAPH Asia 2024 Conference Papers, and

(e) The course project report for the University of Waterloo CS 888 Physics-Based Animation course (Winter 2023), which I authored solely (not a published paper).

For papers (b), (c), and (d), as the sole lead author, Ryusuke Sugimoto proposed and implemented the main algorithms, wrote the initial drafts, and later revised them collaboratively. Across all works, Dr. Toshiya Hachisuka and Dr. Christopher Batty provided supervisory guidance in project direction, technical idea development, and the editing of paper drafts. Paper (a), on the other hand, is a joint first-authored work with a different contribution pattern, as detailed below. Specific additional contributions to each work include:

- For (a), which is joint first-authored, Damien Rioux-Lavoie initially led the project, contributing to core algorithmic design, implementation, analysis, and paper writing. Sugimoto joined later and took the lead, implementing the method on CUDA, improving the evaluation of the vortex stretching term, developing the control variate strategy under Rioux-Lavoie's supervision, and demonstrating robustness against sub-grid-scale and triangle soup obstacles. Tumay Özdemir helped with generating results for traditional methods. Naoharu Shimada contributed to the development of free-slip boundary treatments, a 2D implementation, and an adaptive grid caching

technique. Dr. Derek Nowrouzezahrai, Dr. Batty, and Dr. Hachisuka provided supervisory support. Dr. Hachisuka originally proposed the project idea.

- For (b), Terry Chen implemented the initial three-dimensional results using PBRT. Yiti Jiang collaborated with Dr. Hachisuka on early explorations of the project idea. Both Dr. Hachisuka and Sugimoto independently conceived the initial project concept. Dr. Hachisuka also proposed and implemented several additional components (MCMC, Poisson equation results, 3D results generated with a Monte Carlo rendering system) presented in the paper.
- For (c), there were no other authors involved.
- For (d), Nathan King suggested integrating a Monte Carlo solver within the closest point framework and contributed to the theoretical analysis, the development of the local feature size estimation algorithm with Sugimoto, and paper writing.

In this thesis,

- Chapter 3 largely consists of materials from (b).
- Chapter 4 contains materials from (c) and (e), along with some new developments created solely by me, specifically for this thesis.
- Chapter 5 primarily comprises materials from (a) and (c), with substantial restructuring to present the content from both papers in a consistent manner.
- Chapter 6 largely consists of materials from (d).
- Chapters 1, 2, and 7 contain a combination of sole-authored content by me and curated, paraphrased material from the above works, adapted to the narrative structure of this PhD thesis.

Abstract

This thesis develops novel Monte Carlo methods for solving a wide range of partial differential equations (PDEs) relevant to computer graphics. While traditional discretization-based approaches efficiently compute global solutions, they often require expensive global solves even when only local evaluations are needed, and can struggle with complex or fine-scale geometries. Monte Carlo methods based on the classical Walk on Spheres (WoS) approach [Muller 1956] offer pointwise evaluation with strong geometric robustness, but in practice, their application has been largely limited to interior Dirichlet problems in volumetric domains. We significantly broaden this scope by designing versatile Monte Carlo solvers that handle a diverse set of PDEs and boundary conditions, validated through comprehensive experimental results.

First, we introduce the Walk on Boundary (WoB) method [Sabelfeld 1982, 1991] to graphics. While retaining WoS's advantages, WoB applies to a broader range of second-order linear elliptic and parabolic PDE problems: various boundary conditions (Dirichlet, Neumann, Robin, and mixed) in both interior and exterior domains. Because WoB is based on boundary integral formulations, its structure more closely parallels Monte Carlo rendering than WoS, enabling the application of advanced variance reduction techniques. We present WoB formulations for elliptic Laplace and Poisson equations, time-dependent diffusion problems, and develop a WoB solver for vector-valued Stokes equations. Throughout, we discuss how sampling and variance reduction methods from rendering can be adapted to WoB.

Next, we address the nonlinear Navier–Stokes equations for fluid simulation, whose complexity challenges Monte Carlo techniques. Employing operator splitting, we separate nonlinear terms and solve the remaining linear terms with pointwise Monte Carlo solvers. Recursively applying these solvers with timestepping yields a spatial-discretization-free method. To deal with the resulting exponential computational cost, we also propose cache-based alternatives. Both vorticity- and velocity-based formulations are explored, retaining the advantages of Monte Carlo methods, including geometric robustness and variance reduction, while integrating traditional fluid simulation techniques.

We then propose Projected Walk on Spheres (PWoS), a novel solver for surface PDEs, inspired by the Closest Point Method. PWoS modifies WoS by projecting random walks onto the surface manifold at each step, preserving geometric flexibility and discretization-free, pointwise evaluation. We also adapt a noise filtering technique for WoS to improve PWoS.

Finally, we outline promising future research directions for Monte Carlo PDE solvers in graphics, including concrete proposals to enhance WoB.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my PhD advisors. Toshiya Hachisuka guided me patiently from a point when I was new to physics-based animation and Monte Carlo methods, helping me grow into an independent researcher through just the right balance of support and challenge. Christopher Batty generously shared his expertise, thoughtfully engaging with even my most unconventional questions and ideas, which greatly broadened my understanding both within and beyond my immediate research area. Together, they fostered an open and encouraging environment where I felt empowered to voice my own ideas and shape my research direction. This supportive atmosphere has been instrumental in shaping the researcher I am today, and I believe it is a rare and invaluable experience.

My PhD studies would not have been possible without the patient support of my undergraduate project advisors, Pedro V. Sander, Mingming He, and Jing Liao. From teaching me basic screen projection techniques to guiding me toward my first publication, they generously shared their time and expertise, greatly shaping my learning.

I am deeply grateful to all the collaborators who made my PhD work possible: Terry Chen, Yiti Jiang, Nathan King, Damien Rioux-Lavoie, Tumay Özdemir, Naoharu Shimada, and Derek Nowrouzezahrai.

I would also like to thank all the members of my lab for their fun and insightful discussions. From the Computational Motion Group: Brooke Dolny, Rikin Gurditta, Leo Hanxu, Clara Kim, Nathan King, Daniel Winters, Joel Wretborn, and Sirui Wu. From the Rendering Group: Terry Chen, Ege Ciklabakkal, Maxwell Fang, Haochen Gan, Tianyu Huang, Logan Mosier, Xiaochun Tong, Wenyou Wang, and Weijie Zhou. I am also thankful to the other members of the Computer Graphics Lab at the University of Waterloo for making my time in Waterloo both enjoyable and enriching.

I was fortunate to have the valuable opportunity to be involved in the development of Houdini at SideFX. I am especially grateful to my advisor, Jeff Lait, who taught me the fundamentals of Houdini development and guided me through every stage of the work, and to my senior colleagues—Cristin Barghiel, Mark Elendt, Edward Lam, Ken Taki, Robert Vinluan, and Omar Zarifi—for their guidance and support. I also thank my intern cohort—Kevin Gao, Arshiya Mollazainali, Andrew Muron, Rayhaan Tanweer, David Jacewicz, and Joey Zhou—for their friendship, as well as friendly customers Yunus Balci, Jake Rice, and Fuat Yüksel for their engagement and feedback.

I am also thankful for the many people at Adobe Research who supported me during my internship. My deepest thanks go to Michal Lukáč for leading my project with great

care and for creating such a welcoming and enjoyable atmosphere. I am grateful to Siddhartha Chaudhuri, Kevin Wampler, and Iliyan Georgiev for their valuable advice on my project, and to my group members at different levels—Nathan Carr, Matt Fisher, Difan Liu, Radomír Měch, and Yi Zhou—for their support throughout my time there. I also appreciate my intern cohort—Natalia Pacheco-Tallaj, Christophe Bolduc, Sumit Chaturvedi, Aditya Chetan, Steven Lee, Jiaxin Lu, Sanghyun Son, Vikas Thamizharasan, and Shengxi Wu—for making my time there so enjoyable.

I am grateful as well to the friends I made and to the senior researchers who offered me advice and support at conferences and workshops. In particular, I would like to thank the Eastern Canadian graphics community for providing opportunities to gather and share ideas.

I would also like to thank my committee members for their time and valuable feedback.

I am grateful to all the organizations that supported my studies and research. My graduate studies were partially supported by the David R. Cheriton Graduate Scholarship. The research projects presented in this thesis were partially supported by NSERC Discovery Grants (RGPIN-2021-02524, RGPIN-2020-03918, and RGPIN-2018-05669), CFI-JELF (Grant 40132), and a research grant from Autodesk. This work was enabled in part by computational resources provided by SHARCNET and the Digital Research Alliance of Canada.

Lastly, I would like to thank my family for their constant support throughout my studies.

I acknowledge the use of ChatGPT and Grammarly for language editing assistance during the preparation of this thesis. These tools were only used to improve clarity, grammar, and style, and did not contribute to the technical content, research ideas, or scientific arguments. All intellectual contributions, including problem formulation, method development, and result interpretation, are my own and those of my collaborators.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Partial differential equations (PDEs) play a central role in a wide range of applications in computer graphics and beyond, including, but not limited to, geometry processing, texture generation, and fluid simulation. Since analytical solutions to these PDEs are rarely available, numerical methods are employed to approximate solutions. The popular choices today are finite difference, finite element, finite volume, and boundary element methods. These numerical methods are based on discretizing the domain or the boundary, and then solving a resulting globally coupled matrix equation. However, such discretization-based methods can become tedious, computationally expensive, or even fail when dealing with complex geometries. For instance, generating conforming meshes of the volume [Hu et al. 2018, 2020] (or the surface [De Goes et al. 2022]) can consume substantial time and memory [Sawhney and Crane 2020]. Moreover, even when the solution is only needed at a few specific points or in a small region of interest, these methods still require solving for the solution globally over the entire domain.

To overcome these limitations in the context of geometry processing, Sawhney and Crane [2020] introduced a novel Monte Carlo PDE solver to the graphics community: the *Walk on Spheres* (WoS) method [Muller 1956]. WoS estimates solutions by simulating the effect of continuous random walks through discrete random walks on spheres within the domain. This Monte Carlo approach offers several advantages over conventional methods, including greater flexibility in geometric representation, robustness, trivial parallelism, and the ability to estimate the solution *pointwise* without a global solve. This shift toward Monte Carlo PDE solvers parallels the transition in computer graphics from radiosity-

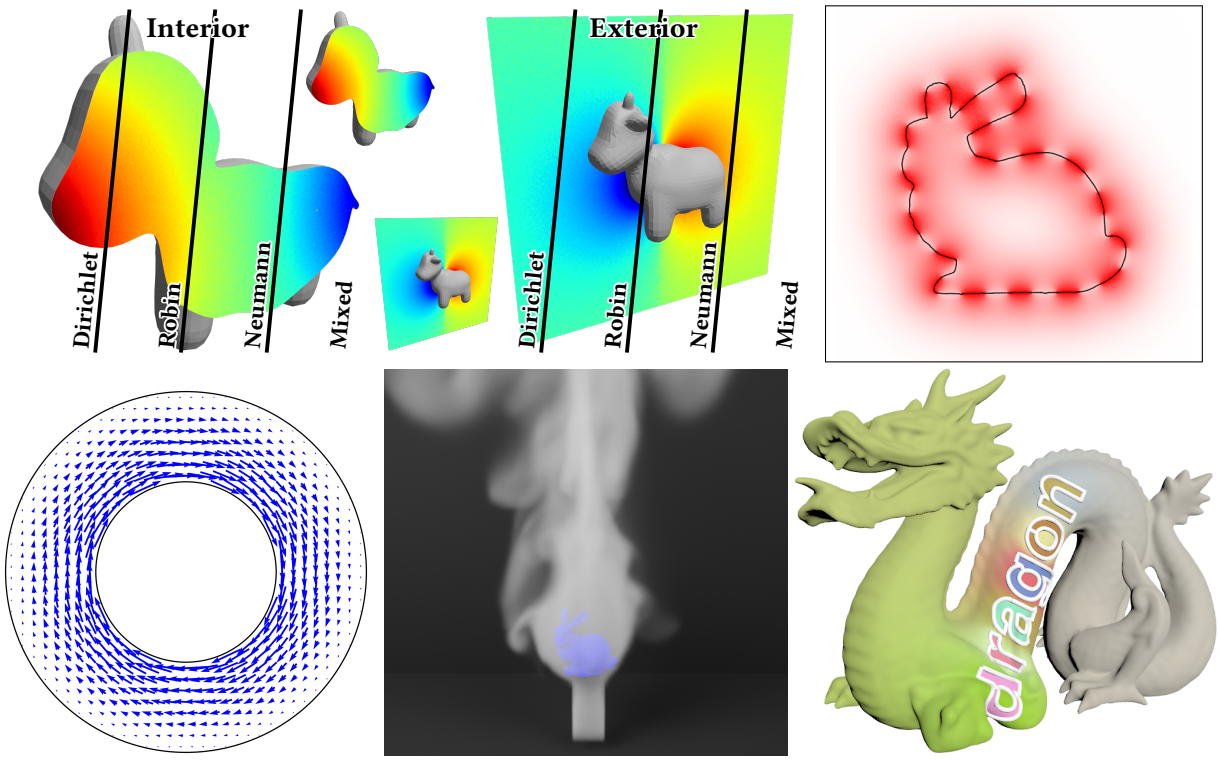


Figure 1.1: This thesis explores general-purpose Monte Carlo PDE solvers. We discuss the Walk on Boundary method for linear PDEs, which supports various boundary conditions within both interior and exterior domains (top-left), and its application to time-dependent diffusion (top-right) and vector-valued Stokes (bottom-left) equations. We also develop Monte Carlo PDE solvers for the Navier–Stokes equations (bottom-middle), and present the Projected Walk on Spheres method for surface PDEs (bottom-right).

based light transport simulation [Cohen and Greenberg 1985; Goral et al. 1984; Nishita and Nakamae 1985] to Monte Carlo rendering methods [Cook 1986; Cook et al. 1984; Kajiya 1986; Pharr et al. 2018]. Sawhney and Crane [2020] and others indeed showed that the connection between WoS and Monte Carlo ray tracing enables various techniques from Monte Carlo ray tracing to be modified and adapted to WoS, such as in recent work by Qi et al. [2022] on bidirectional WoS.

PDE problems encompass a variety of boundary conditions and equations. However, the basic WoS algorithm is limited to linear PDEs posed on volumetric, interior domains with Dirichlet boundary conditions. With this context in mind, this thesis investigates general techniques and applications of Monte Carlo methods for a range of PDE problems we encounter in the context of graphics.

1.2 Problem Scope and Approach

In this thesis, we aim to advance the design of *general-purpose* Monte Carlo PDE solvers, primarily targeting graphics applications (Fig. 1.1). After reviewing relevant background and related work (Chapter 2), we present Monte Carlo PDE solvers addressing a broad range of problems, including various linear PDEs with various boundary conditions (Chapters 3 and 4), the nonlinear Navier–Stokes equations (Chapter 5), and surface PDEs (Chapter 6). We conclude by outlining promising future research directions (Chapter 7).

In Chapter 3, we introduce the *Walk on Boundary* (WoB) method, originally proposed by Sabelfeld [1982, 1991], to the computer graphics community, focusing on the Laplace and Poisson equations. Like WoS, WoB is a pointwise Monte Carlo method. While WoS simulates Brownian motion within the domain, WoB is derived directly from *boundary integral equations* (BIEs) obtained by reformulating the PDE. This reformulation leads to an algorithm that traces rays *between points on the boundary* (Fig. 1.2), analogous to the Monte Carlo ray tracing algorithm for the rendering equation [Kajiya 1986]. This boundary-focused formulation fundamentally differs from the domain interior random walks used by WoS.

Although arguably less well-known than WoS, WoB shares with WoS many advantages over spatial discretization-based methods. WoB also offers several unique advantages, which we explore and demonstrate:

- **Generality:** WoB naturally handles boundary value problems for which WoS is inefficient or inapplicable. WoS struggles with Neumann or Robin boundary conditions,

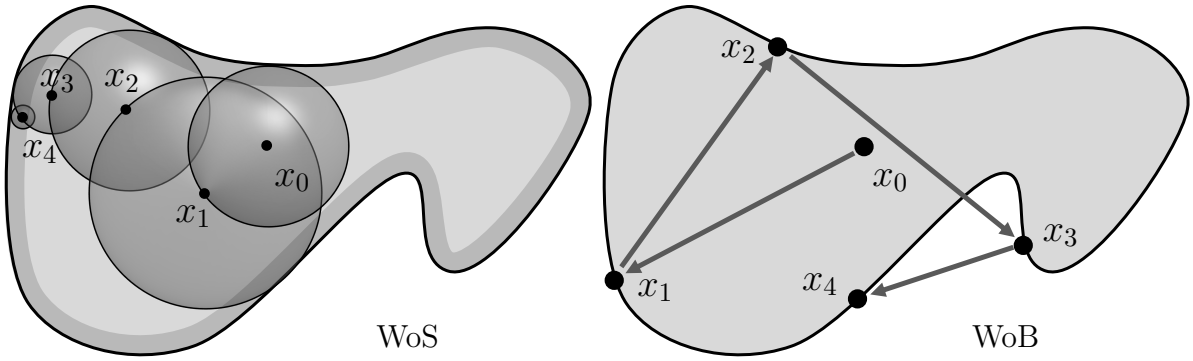


Figure 1.2: While WoS takes random walks on spheres, WoB takes random walks on the boundary to compute a sample contribution.

especially on non-convex domains [Simonov 2008, 2017], and only very recent concurrent work [Miller et al. 2024b; Sawhney et al. 2023] addresses this efficiently. Exterior domain problems require complicated transformations in WoS [Nabizadeh et al. 2021], while WoB seamlessly handles interior and exterior problems with various boundary conditions, without changing the core formulation (see Fig. 1.1 top left).

- **Accuracy:** WoB can accurately estimate solutions right *on* or *near* the boundary, whereas WoS's random walks must terminate in an ϵ-shell, introducing bias and errors. Fig. 1.3 illustrates these differences. Solutions on the boundary are relevant for Neumann and Robin problems, since the boundary values are unknown, and they are the main quantity of interest for certain cases [Da et al. 2016; Sugimoto et al. 2022].

- **Similarity to Monte Carlo ray tracing:** WoB samples points *on* the boundary, making it algorithmically and mathematically closer to Monte Carlo ray tracing than WoS, which operates inside the domain with closest point queries. This similarity allows WoB to leverage existing ray tracing frameworks and to adapt advanced rendering techniques more readily.

In Chapter 4, we extend WoB to a broader class of PDEs beyond the Laplace and Poisson equations. Specifically, we introduce WoB for the diffusion equation [Sabelfeld 1991] and develop a new WoB formulation for the Stokes equations, which govern viscous, incompressible fluid flow. Diffusion equation introduces time dependence, requiring temporal variables to be incorporated into the boundary walks, while the Stokes equations are vector-valued, necessitating an extension of WoB from scalar to vector- and matrix-valued quantities. In both cases, only modest modifications to the basic WoB estimators

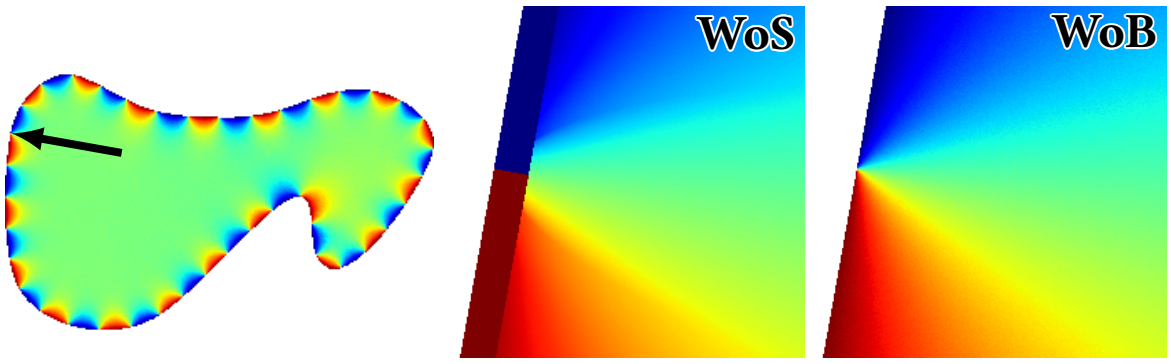


Figure 1.3: Unlike WoS, WoB does not introduce any errors associated with the ϵ-shell that WoS uses to terminate their random walk. The indicated region (left) displays visible banding artifacts with WoS (middle), where WoB presents no such error (right).

for the Laplace and Poisson equations are necessary. We demonstrate that the core advantages of WoB—pointwise solution evaluation and compatibility with variance reduction techniques—remain intact even for these more complex equations.

In Chapter 5, we explore the application of Monte Carlo PDE solvers to the nonlinear Navier–Stokes equations, which govern the behavior of fluids. These equations are considerably more complex than linear PDEs to which Monte Carlo PDE solvers are typically applied. This complexity is due in part to the presence of a nonlinear advection term, along with several other terms. We address this challenge using the operator splitting method [Park and Kim 2005; Stam 1999]. This method decomposes the Navier–Stokes equations—either in their vorticity or velocity formulation—into simpler subproblems. Within each timestep, we solve these subproblems as substeps sequentially, isolating the nonlinear advection from the remaining terms. We can then handle the latter using Monte Carlo PDE solvers, such as WoB or WoS. We design all substep solvers as pointwise estimators. By recursively applying these estimators, we obtain a spatial-discretization-free solver; given a query point in space and time, this solver can directly estimate the solution of the Navier–Stokes equations, mirroring the capabilities of existing Monte Carlo PDE solvers. However, this fully pointwise recursive formulation incurs exponential computational cost with respect to the number of timesteps. As a practical alternative, we also introduce discrete caching into the formulation. While caching sacrifices the fully discretization-free property, it preserves other key advantages of Monte Carlo PDE solvers, including the ability to handle complex geometries. Moreover, cached data can be exploited to improve Monte Carlo sampling for variance reduction. Overall, this work represents a first step toward extending Monte Carlo PDE solvers to more complex, nonlinear PDEs

for practical applications.

The methods discussed thus far are primarily aimed at volumetric PDEs, i.e., equations whose solutions are defined on an n-dimensional manifold embedded in n-dimensional space. In Chapter 6, we lift this restriction by introducing ideas from the Closest Point Method (CPM) [King et al. 2024; März and Macdonald 2012; Ruuth and Merriman 2008], which solves surface PDEs using volumetric differential operators. The resulting *Projected Walk on Spheres* (PWoS) method for *surface PDEs* takes random walks on spheres in the neighborhood of the surface, followed by a *projection* of the sample point onto the surface. This simple modification extends WoS to surface PDEs, including PDEs defined on mixed-codimensional geometries, while retaining some of the key advantages: independence from specific surface discretizations or resolutions, no need for conforming meshes, trivial parallelism, and pointwise evaluation. We also investigate extending the mean value filtering method [Bakbouk and Peers 2023] to reduce noise in PWoS solutions when the surface is discretized.

Together, these contributions explore the design of general-purpose Monte Carlo PDE solvers. While motivated by computer graphics, the methods have potential impact across scientific and engineering fields. In Chapter 7, we discuss several promising avenues for improving applicability, WoB's efficiency, and hybrid integration with traditional discretization-based solvers.

In summary, this thesis makes the following key contributions:

- Introduction of the Walk on Boundary (WoB) method to computer graphics for the Laplace, Poisson, and diffusion equations, providing a unified framework capable of handling diverse boundary conditions (Chapters 3 and 4).
- First application of WoB to the Stokes equations, extending the formulation to handle the vector-valued PDE governing viscous incompressible flow (Chapter 4).
- Development of Monte Carlo PDE solvers for the Navier–Stokes equations using operator splitting in both vorticity and velocity formulations, enabling pointwise or cached evaluations without global discretization (Chapter 5).
- Formulation of Projected Walk on Spheres (PWoS) for surface PDEs, generalizing WoS with the Closest Point Method to solve PDEs defined on surfaces without requiring conforming meshes (Chapter 6).
- Bridging the above Monte Carlo PDE solvers with Monte Carlo rendering techniques, adapting advanced sampling and variance-reduction strategies to PDE contexts (Chapters 3, 4, 5, and 6).

- Identification of promising future research directions, including concrete proposals to enhance WoB's efficiency, along with ideas for extending applicability and integrating with discretization-based methods (Chapter 7).

Chapter 2

Background and Related Work

2.1 Monte Carlo Methods in Graphics

Monte Carlo methods have been successfully applied to a diversity of problems in, e.g., statistical inference, simulation, and integration [Metropolis and Ulam 1949]. In graphics, Monte Carlo methods have been used most extensively in physically-based light transport simulation, where they outpaced traditional finite element methods to treat challenging radiometric effects with increasingly complex geometric and reflectance models [Kajiya 1986; Pharr et al. 2018]. In 2020, Sawhney and Crane [2020] proposed the application of Monte Carlo methods to geometry processing to further evidence its broader applicability in graphics.

2.1.1 Monte Carlo Ray Tracing for Rendering

Cook et al. [1984] introduced the use of Monte Carlo methods in rendering to simulate effects such as soft shadows, motion blur, and depth of field. Following the formalization of the now foundational *rendering equation* [Cook 1986; Kajiya 1986], Monte Carlo rendering has become a dominant approach. At the time of its introduction, Monte Carlo rendering was considered too computationally expensive to be practical. Finite element radiosity methods [Cohen and Greenberg 1985; Goral et al. 1984; Nishita and Nakamae 1985] were the prevailing standard for solving radiative transfer problems. However, with advances in sampling techniques [Müller et al. 2017; Veach and Guibas 1995, 1997], GPU hardware acceleration [Parker et al. 2010; Purcell et al. 2002], and machine learning-based

denoisers [Kalantari et al. 2015], Monte Carlo rendering has become viable even in real-time contexts [Bitterli et al. 2020]. Monte Carlo rendering has evolved to capture increasingly complex radiometric phenomena while scaling effectively with the growing complexity of virtual environments [Pharr et al. 2018]. Today, modern industrial-caliber renderers employ Monte Carlo rendering [Pharr 2018] along with specialized variance reduction techniques to achieve practical efficiency. The long-term goal of the research presented in this thesis is to extend such Monte Carlo techniques to broader problem domains in computer graphics beyond rendering, and potentially to scientific computing more generally.

2.1.2 Monte Carlo Integration Basics

The Monte Carlo methods explored in this thesis are based on reformulating a given PDE as integration problems, which we then solve using Monte Carlo integration. As such, Monte Carlo integration lies at the core of the methodologies, and we briefly review its basic concepts here. For a more comprehensive review, see Pharr et al. [2018] in the context of graphics, or Owen [2013] for a general treatment.

We consider the problem of estimating an n-dimensional integral over a domain Ω, for example:

$$I = \int_\Omega f(\mathbf{x}) \, \mathrm{d}\mathbf{x}. \tag{2.1}$$

Deterministic Quadrature

In general, this integral cannot be evaluated analytically, necessitating numerical techniques. One classical approach is to use a deterministic *quadrature* rule. For example, we can subdivide the domain into uniform hypercubes, evaluate the integrand at the center of each subdomain, and average the results. Multiplying the average by the hypervolume of each subdomain yields an estimate of Eq. 2.1. More accurate results can be obtained using non-uniform sample points, for instance via Gaussian quadrature. While quadrature is conceptually simple and deterministic, its convergence rate deteriorates rapidly as the dimension increases. In particular, the rate typically scales as $O(N^{-1/n})$, where N is the number of evaluation points and n is the dimension. This phenomenon is often referred to as the *curse of dimensionality*.

Monte Carlo integration

Monte Carlo integration offers an alternative that overcomes some of these limitations. In Monte Carlo integration, we generate N typically independent and identically distributed (i.i.d.) samples $\{\mathbf{x}_i\}_{i=1}^N$ from a *probability density function* (PDF) $p(\mathbf{x})$ defined over Ω. We then estimate the integral as:

$$\hat{I}_N = \frac{1}{N}\sum_{i=1}^{N}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}. \tag{2.2}$$

Here, the hat notation $\hat{I}_N$ indicates that this is an estimator of the true integral. Note that the PDF $p(\mathbf{x})$ is normalized, i.e., it integrates to 1 over Ω and it must be nonzero whenever $\mathbf{x}$ is nonzero within Ω. In practice, we may say we generate samples according to an unnormalized function, with the understanding that the PDF is the corresponding normalized function.

To discuss the properties of the Monte Carlo estimator in Eq. 2.2, we introduce several key concepts. First, we consider the *expectation* of $\hat{I}_N$. If $\mathbb{E}[\hat{I}_N] = I$, we say the estimator is *unbiased*. Otherwise, it is *biased*, and the difference $\mathbb{E}[\hat{I}_N] - I$ is referred to as the *bias*. We also examine the behavior of the estimator as the number of samples increases. If $\hat{I}_N \to I$ as $N \to \infty$, we say the estimator is *consistent*.

However, bias and consistency alone do not provide a complete picture, as in practice we can only afford a finite number of samples. The *variance* of the estimator, $\mathrm{Var}[\hat{I}_N] = \mathbb{E}\left[(\hat{I}_N - \mathbb{E}[\hat{I}_N])^2\right]$, quantifies the level of noise in the estimate due to sampling randomness. A lower variance generally results in a less noisy result and more reliable results, as we can see from an associated confidence interval analysis. When the estimator is unbiased, the variance is equal to the *mean squared error* (MSE). In general, we define MSE to be $\mathrm{MSE}[\hat{I}_N] = \mathrm{Var}[\hat{I}_N] + \left(\mathbb{E}[\hat{I}_N] - I\right)^2$.

As the name suggests, the MSE quantifies the expected *squared* error. To understand the scale of the error itself, we often consider the square root of the MSE. For an unbiased estimator, the MSE reduces to the variance, which decreases proportionally to $1/N$. Consequently, the standard deviation (i.e., the square root of the variance) decreases at the rate of $O(1/\sqrt{N})$, independent of the dimension of the integral. This dimension-independence is a key advantage of Monte Carlo methods over deterministic quadrature techniques, whose convergence rates typically deteriorate rapidly in high-dimensional settings.

Focusing on the MSE, we observe that the source of error in Monte Carlo estimators arises from two components: variance and bias. Even with an unbiased estimator, a large

variance can lead to significant noise in the estimate, resulting in large errors. Conversely, an estimator with low or zero variance can still produce large errors if it has substantial bias. In practice, these two quantities often exhibit a trade-off relationship: reducing variance may require accepting some bias, and reducing bias may increase variance. The primary goal in designing an estimator in our application is to minimize the overall error given a fixed computational budget. However, in certain applications, such as real-time rendering with temporal information reuse, we apply Monte Carlo estimators repeatedly. In these situations, it can become important to use unbiased estimators to prevent error accumulation that can degrade the final result over time.

Variance reduction

To improve the efficiency of Monte Carlo integration, a variety of variance reduction techniques have been developed and applied to different problems. Two representative approaches are *importance sampling* and *control variates*.

With importance sampling, the idea is to concentrate samples in regions of the domain that contribute more significantly to the integral, thereby reducing the estimator's variance. In Eq. 2.2, the most naive strategy would be to sample uniformly over the domain. For simplicity, assume the integrand $f(\mathbf{x})$ is a nonnegative scalar function. If we could instead sample from a PDF $p(\mathbf{x})$ proportional to $f(\mathbf{x})$, then the resulting estimator would have *zero variance*—the Monte Carlo estimate would give the exact result even with a single sample. Of course, such a sampling strategy is not practical in general, as being able to sample exactly from such a $p(\mathbf{x})$ implies we already know the integral I. Nevertheless, this ideal case serves as a useful guide, and we aim to set $p(\mathbf{x})$ so it is roughly proportional to the integrand in some sense. In practice, the integrand is often a product of multiple functions, and we may design $p(\mathbf{x})$ to match one of them, and we can expect that it yields a lower-variance estimator.

In the method of (difference) control variates, we reduce variance by leveraging knowledge of an auxiliary function with a known integral. Suppose we have a function $g(\mathbf{x})$ such that its integral $I_g = \int_\Omega g(\mathbf{x})\,\mathrm{d}\mathbf{x}$ is known analytically. We then rewrite the original integral Eq. 2.1 as

$$I = \int_\Omega \{f(\mathbf{x}) - g(\mathbf{x})\}\,\mathrm{d}\mathbf{x} + I_g, \tag{2.3}$$

and apply Monte Carlo integration to the difference $f(\mathbf{x}) - g(\mathbf{x})$. If we can have a Monte Carlo estimator for the integral of this difference that has lower variance than the original integral, we achieve variance reduction.

These are just two of the simplest and widely used variance reduction techniques. In practice, many problem-specific strategies exist. In the methods we discuss throughout this thesis, we will utilize several variance reduction techniques, including the ones introduced here.

2.2 Partial Differential Equations and Boundary Integral Equations

PDEs are fundamental tools in computer graphics and many areas of scientific computing. The WoB solver introduced in this thesis reformulates PDEs as boundary integral equations (BIEs) and applies Monte Carlo integration techniques to solve them. In this section, we review the core concepts underlying this approach. The book by Sabelfeld and Simonov [1994] and notes by Pechstein [2013] discuss the details of PDEs and BIEs summarized here.

2.2.1 Partial Differential Equations

PDEs can model a wide range of phenomena in physics and geometry. This thesis primarily focuses on linear second-order PDEs. To introduce the core concepts, we begin with the Poisson equation—a representative scalar-valued elliptic PDE that is time-independent. The ideas discussed in this section naturally extend to a broader class of problems.

In particular, the formulation generalizes to other elliptic PDEs, such as the Helmholtz and screened Poisson equations. Later in the thesis, we also discuss how the WoB solver can be extended to handle time-dependent (parabolic) and vector-valued PDEs, along with the necessary modifications to both the PDE and BIE formulations (Chapter 4). Also, building on solvers for linear PDEs, we show that it is possible to develop methods for the nonlinear Navier–Stokes equations as well (Chapter 5). The PDEs we discuss here are written for volumetric PDEs, and we will discuss a solver for highly related surface PDEs in Chapter 6, too.

The Poisson equation, a representative linear second-order PDE, is commonly given as

$$\nabla^2 v(\mathbf{x}) = \bar{b}(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \Omega \tag{2.4}$$

where Ω is a closed domain in $\mathbb{R}^2$ or $\mathbb{R}^3$, ∇^2 is the Laplacian operator defined as the divergence of gradient, $v(\mathbf{x})$ is the unknown field, and $\bar{b}(\mathbf{x})$ is a known source function

defined inside the domain. The Poisson equation with $\bar{b}(\mathbf{x}) = 0$ is also called the Laplace equation. In Section 3.2.5, we will consider exterior problems where the domain is $\mathbb{R}^2 \backslash \Omega$ or $\mathbb{R}^3 \backslash \Omega$. Let us denote the boundary of Ω by $\Gamma = \partial\Omega$; note that Γ is not included in Ω. The *boundary value problems* in this thesis concern solving for $v(\mathbf{x})$ in Eq. 2.4 with boundary conditions

$$\begin{aligned} v(\mathbf{x}) &= \overline{u}_D(\mathbf{x}) && \text{for } \mathbf{x} \in \Gamma_D \subseteq \Gamma, \\ \frac{\partial v}{\partial \mathbf{n}}(\mathbf{x}) &= \overline{q}_N(\mathbf{x}) && \text{for } \mathbf{x} \in \Gamma_N \subseteq \Gamma, \\ \frac{\partial v}{\partial \mathbf{n}}(\mathbf{x}) + \overline{\alpha}(\mathbf{x}) v(\mathbf{x}) &= \overline{g}_R(\mathbf{x}) && \text{for } \mathbf{x} \in \Gamma_R \subseteq \Gamma, \end{aligned} \tag{2.5}$$

where the normal derivative $\frac{\partial v}{\partial \mathbf{n}}(\mathbf{x}) = \mathbf{n}(\mathbf{x}) \cdot \nabla v(\mathbf{x})$, the weight $\overline{\alpha}(\mathbf{x}) \neq 0$, and a bar indicates that the function is given. Any point $\mathbf{x} \in \Gamma$ belongs to strictly one of Γ_D, Γ_N, or Γ_R. When $\Gamma_D = \Gamma$, $\Gamma_N = \Gamma$, or $\Gamma_R = \Gamma$, we call the problem a Dirichlet, Neumann, or Robin problem, respectively. Otherwise, we call it a mixed boundary problem.

The analytical solution to the Poisson equation is typically not known, and we therefore resort to numerical methods such as the finite difference [Thomas 1995], finite element [Reddy 1993], or finite volume method [LeVeque 2002]. By discretizing the domain, these methods exploit local relationships in the PDE and construct a matrix system. In contrast, the WoB approach avoids any discretization of the domain (or boundary) by leveraging the integral equation formulation.

One can always convert the Poisson equation for $v(\mathbf{x})$ (Equations 2.4 and 2.5) into the Laplace equation for $u(\mathbf{x})$ using a relation $u(\mathbf{x}) = v(\mathbf{x}) + V_0(\mathbf{x})$ as follows. Let u be a function satisfying the Laplace equation

$$\begin{aligned} \nabla^2 u(\mathbf{x}) &= 0 && \text{for } \mathbf{x} \in \Omega, \\ u(\mathbf{x}) &= \overline{u}_D(\mathbf{x}) + V_0(\mathbf{x}) && \text{for } \mathbf{x} \in \Gamma_D, \\ \frac{\partial u}{\partial \mathbf{n}}(\mathbf{x}) &= \overline{q}_N(\mathbf{x}) + \frac{\partial V_0}{\partial \mathbf{n}}(\mathbf{x}) && \text{for } \mathbf{x} \in \Gamma_N, \\ \frac{\partial u}{\partial \mathbf{n}}(\mathbf{x}) + \overline{\alpha}(\mathbf{x}) u(\mathbf{x}) &= \overline{g}_R(\mathbf{x}) + \frac{\partial V_0}{\partial \mathbf{n}}(\mathbf{x}) + \overline{\alpha}(\mathbf{x}) V_0(\mathbf{x}) && \text{for } \mathbf{x} \in \Gamma_R, \end{aligned} \tag{2.6}$$

where $V_0(\mathbf{x}) = \int_\Omega G(\mathbf{x}, \mathbf{y}) \bar{b}(\mathbf{y}) \, \mathrm{d}\mathbf{y}$ and $G(\mathbf{x}, \mathbf{y})$ is the fundamental solution as we define in the next section. One can confirm that $v(\mathbf{x}) = u(\mathbf{x}) - V_0(\mathbf{x})$ satisfies the Poisson equation by noting that $\nabla^2 V_0(\mathbf{x}) = -\bar{b}(\mathbf{x})$ for $\mathbf{x} \in \Omega$. Thus, in Chapter 3, we will focus on solutions to the Laplace equation since it is easy to handle the additional contribution from $V_0(\mathbf{x})$.

2.2.2 Fundamental Solutions

The fundamental solution $G(\mathbf{x}, \mathbf{y})$ for the Laplace operator is a solution to the Poisson equation in an infinite domain with a (negative) Dirac delta source: $\nabla^2 G(\mathbf{x}, \mathbf{y}) + \delta(\mathbf{x} - \mathbf{y}) = 0$. Note that different texts may use different sign conventions. In 3D, we have $G(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi r}$ and its derivatives are

$$\frac{\partial G}{\partial \mathbf{x}_\mathsf{k}}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{r} \cdot \mathbf{e}_\mathsf{k}}{4\pi r^3}, \quad \frac{\partial G}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}, \mathbf{y}) = -\frac{\mathbf{r} \cdot \mathbf{n}_\mathbf{y}}{4\pi r^3}, \quad \frac{\partial G}{\partial \mathbf{n}_\mathbf{x}}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{r} \cdot \mathbf{n}_\mathbf{x}}{4\pi r^3},$$
$$\frac{\partial^2 G}{\partial \mathbf{x}_\mathsf{k} \partial \mathbf{n}_\mathbf{y}}(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi} \left[\frac{\mathbf{n}_\mathbf{y} \cdot \mathbf{e}_\mathsf{k}}{r^3} - 3 \frac{(\mathbf{r} \cdot \mathbf{n}_\mathbf{y})(\mathbf{r} \cdot \mathbf{e}_\mathsf{k})}{r^5} \right].$$

In 2D, we have $G(\mathbf{x}, \mathbf{y}) = -\frac{1}{2\pi} \log r$ and its derivatives are

$$\frac{\partial G}{\partial \mathbf{x}_\mathsf{k}}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{r} \cdot \mathbf{e}_\mathsf{k}}{2\pi r^2}, \quad \frac{\partial G}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}, \mathbf{y}) = -\frac{\mathbf{r} \cdot \mathbf{n}_\mathbf{y}}{2\pi r^2}, \quad \frac{\partial G}{\partial \mathbf{n}_\mathbf{x}}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{r} \cdot \mathbf{n}_\mathbf{x}}{2\pi r^2},$$
$$\frac{\partial^2 G}{\partial \mathbf{x}_\mathsf{k} \partial \mathbf{n}_\mathbf{y}}(\mathbf{x}, \mathbf{y}) = \frac{1}{2\pi} \left[\frac{\mathbf{n}_\mathbf{y} \cdot \mathbf{e}_\mathsf{k}}{r^2} - 2 \frac{(\mathbf{r} \cdot \mathbf{n}_\mathbf{y})(\mathbf{r} \cdot \mathbf{e}_\mathsf{k})}{r^4} \right].$$

where $\mathbf{r} = \mathbf{y} - \mathbf{x}$, $r = \|\mathbf{r}\|$, and $\mathbf{e}_k$ is the k-th basis vector. Note that the fundamental solutions and their derivatives exhibit singular behavior: as $r \to 0$, these functions become unbounded. Consequently, special care is required when handling them, particularly near singularities.

For an infinite domain problem ($\Omega = \mathbb{R}^d$), we can convolve the fundamental solution and the source term to get the solution to Eq. 2.4:

$$v(\mathbf{x}) = -\int_{\mathbb{R}^d} G(\mathbf{x}, \mathbf{y}) \bar{b}(\mathbf{y}) \, \mathrm{d}\mathbf{y}, \tag{2.7}$$

which we can evaluate with quadrature or Monte Carlo integration.

Beyond isolating the contribution of the source term in the Poisson equation and solving PDEs in an infinite domain, fundamental solutions allow us to reformulate PDEs as BIEs for problems with boundaries, as we discuss next.

2.2.3 Boundary Integral Equations

WoB builds upon BIE formulations. Such formulations are applicable to many second-order linear elliptic PDEs [Clements 2004] and some other PDEs [Liu et al. 2012], but we again focus on the BIE formulation for the Poisson equation here.

Direct Boundary Integral Equations

The basic idea of boundary integral formulations is to define the solution based on integrals *only of boundary values*. They can actually take several different forms. We first explain one such formulation called a *direct* BIE formulation. As the name suggests, direct BIEs describe the relationship between the solution values in the interior or on the boundary *directly*, and are derived based on Green's third identity. One common such direct BIE is given as

$$c(\mathbf{x})u(\mathbf{x}) = -\int_\Gamma \frac{\partial G}{\partial \mathbf{n_y}}(\mathbf{x},\mathbf{y})u(\mathbf{y})\,\mathrm{d}\mathbf{y} + \int_\Gamma G(\mathbf{x},\mathbf{y})\frac{\partial u}{\partial \mathbf{n}}(\mathbf{y})\,\mathrm{d}\mathbf{y} \tag{2.8}$$

for $\mathbf{x} \in \Omega \cup \Gamma$, where $c(\mathbf{x})$ is an integral free term that evaluates to 1 if $\mathbf{x} \in \Omega$, and to $\frac{1}{2}$ if $\mathbf{x} \in \Gamma$ when the boundary is smooth in the sense of Lyapunov. We consider only smooth surfaces in the following, but the extension to non-smooth surfaces is straightforward. Note that polygonal boundaries do not violate the smoothness assumption, unless $\mathbf{x}$ is evaluated *exactly* on a vertex of the polygon.

Eq. 2.8 says that the solution $u(\mathbf{x})$ within the domain and on the boundary satisfies this integral equation involving only boundary integrals. The boundary conditions alone do not directly provide $u(\mathbf{y})$ and $\frac{\partial u}{\partial \mathbf{n}}(\mathbf{y})$ everywhere along Γ, so one must solve for such unknown boundary values first to evaluate the solution $u(\mathbf{x})$ inside of the domain.

One can also derive a BIE for the directional derivative of the solution by taking the directional derivative of Eq. 2.8:

$$\frac{\partial u}{\partial \mathbf{x}_k}(\mathbf{x}) = -\int_\Gamma \frac{\partial^2 G}{\partial \mathbf{x}_k \partial \mathbf{n_y}}(\mathbf{x},\mathbf{y})u(\mathbf{y})\,\mathrm{d}\mathbf{y} + \int_\Gamma \frac{\partial G}{\partial \mathbf{x}_k}(\mathbf{x},\mathbf{y})\frac{\partial u}{\partial \mathbf{n}}(\mathbf{y})\,\mathrm{d}\mathbf{y}, \tag{2.9}$$

for $\mathbf{x} \in \Omega$. We denote the first order derivative with respect to the k-th direction by $\partial u/\partial \mathbf{x}_k$. This equation is valid only in the interior of the domain, so we omitted the integral free term $c(\mathbf{x}) = 1$.

Indirect Boundary Integral Equations

In contrast to direct BIEs, an *indirect* BIE describes the relationship between an unknown *source density function* on the boundary and the known boundary values, and expresess the solution only *indirectly* based on the source density function. There are two types of indirect formulations, derived from potential theory.

Single layer potential The solution $u(\mathbf{x})$ to the Laplace equation can be expressed in the form of a *single layer potential* given by

$$u(\mathbf{x}) = \int_\Gamma G(\mathbf{x},\mathbf{y})\mu(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Omega \cup \Gamma, \tag{2.10}$$

where the unknown source density function μ corresponds to the jump of the normal derivative of u across the boundary. In short, Eq. 2.10 expresses the solution inside the domain (and on the boundary) in terms of monopole sources, distributed over the boundary, which decay according to the fundamental solution $G(\mathbf{x},\mathbf{y})$. Taking the directional derivative of Eq. 2.10 and taking the limit to the boundary gives an integral equation for $\mu(\mathbf{x})$:

$$\frac{\partial u}{\partial \mathbf{n}}(\mathbf{x}) = c(\mathbf{x})\mu(\mathbf{x}) + \int_\Gamma \frac{\partial G}{\partial \mathbf{n_x}}(\mathbf{x},\mathbf{y})\mu(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Gamma. \tag{2.11}$$

Similarly, the directional derivative at any interior point $\mathbf{x}$ is

$$\frac{\partial u}{\partial \mathbf{x}_k}(\mathbf{x}) = \int_\Gamma \frac{\partial G}{\partial \mathbf{x}_k}(\mathbf{x},\mathbf{y})\mu(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Omega. \tag{2.12}$$

This equation is invalid exactly on the boundary because of jump discontinuities across Γ.

Double layer potential An alternative is a *double layer potential* which uses dipole source on the boundary:

$$u(\mathbf{x}) = -\int_\Gamma \frac{\partial G}{\partial \mathbf{n_y}}(\mathbf{x},\mathbf{y})\nu(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Omega \tag{2.13}$$

where ν is an unknown source density function, corresponding to the jump of the solution across the boundary. In the limit as $\mathbf{x} \to \Gamma$, one finds an integral equation for $\nu(\mathbf{x})$:

$$u(\mathbf{x}) = [1 - c(\mathbf{x})]\nu(\mathbf{x}) - \int_\Gamma \frac{\partial G}{\partial \mathbf{n_y}}(\mathbf{x},\mathbf{y})\nu(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Gamma. \tag{2.14}$$

The normal derivative can be computed with

$$\frac{\partial u}{\partial \mathbf{n_x}}(\mathbf{x}) = -\frac{\partial}{\partial \mathbf{n_x}} \int_\Gamma \frac{\partial G}{\partial \mathbf{n_y}}(\mathbf{x},\mathbf{y})\nu(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Gamma, \tag{2.15}$$

and the directional derivative can be computed with

$$\frac{\partial u}{\partial \mathbf{x}_k}(\mathbf{x}) = -\int_\Gamma \frac{\partial^2 G}{\partial \mathbf{x}_k \partial \mathbf{n_y}}(\mathbf{x},\mathbf{y})\nu(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Omega. \tag{2.16}$$

In Chapter 3, by adopting different formulations for different problems, we will develop practical numerical WoB methods.

2.2.4 Boundary Element Method and Method of Fundamental Solutions

In this thesis, we explore the WoB method for solving the BIEs described above. While our approach is based on Monte Carlo techniques, BIEs have traditionally been tackled using boundary discretization-based methods, such as the *boundary element method* (BEM) [Liu et al. 2012] and the *method of fundamental solutions* (MFS) [Cheng and Hong 2020].

BEM solves a BIE by approximating the unknown boundary density functions using a discrete basis. For example, in 2D, one can discretize the boundary into segments and represent the boundary density functions using piecewise linear functions over these segments. A key advantage of BEM is the reduction in problem dimensionality, as only the boundary requires discretization. This feature also makes it well-suited for problems in infinite domains (e.g., for sound simulations). However, BEM leads to a large, dense linear system that one must solve to obtain the boundary values. In addition, accurately evaluating integrals involving singular kernels becomes tedious.

The MFS also relies on the same underlying BIEs but avoids exact boundary discretization by placing point sources (fundamental solutions) slightly outside the domain. While this can simplify implementation in some settings, the method struggles with generality, particularly for complex boundary shapes, because the placement of the point sources can become problematic. Moreover, it still requires solving a dense linear system.

In computer graphics, these discretization-based BIE solvers and related techniques have been applied to various problems, including diffusion curves [Bang et al. 2023; Chen et al. 2024; Ilbery et al. 2013; Sun et al. 2014, 2012; van de Gronde 2010; Xie et al. 2014], linear elasticity [Hahn and Wojtan 2015, 2016; James and Pai 1999, 2003; Sugimoto et al. 2022; Zhu et al. 2015], liquid simulation [Da et al. 2016; Huang and Michels 2020; Huang et al. 2021; Keeler and Bridson 2014; Ni et al. 2024], sound simulation [Chadwick et al. 2009; James et al. 2006; Umetani et al. 2016; Zheng and James 2009, 2010], and several geometry processing tasks [Chen et al. 2023; Levi and Weber 2016; Lipman et al. 2008; Solomon et al. 2017; Wang et al. 2013].

In contrast to these approaches, the WoB method avoids explicit discretization and dense linear solves, offering a flexible and potentially more scalable alternative depending on problem settings. This relationship between WoB and these methods is analogous to that between rendering techniques based on Monte Carlo methods and the radiosity method [Cohen and Greenberg 1985; Goral et al. 1984; Nishita and Nakamae 1985] for the rendering equation. These traditional BIE-based techniques also suggest other potential application domains of WoB.

2.3 Monte Carlo PDE Solvers — Walk on Spheres

Recently in computer graphics, Sawhney and Crane [2020] introduced a Monte Carlo framework for solving PDEs in geometry processing. Their work brought the WoS method [Delaurentis and Romero 1990; Elepov and Mikhailov 1969; Muller 1956] to the graphics community, highlighting its advantages over discretization-based methods, including greater flexibility in geometric representation, robustness, trivial parallelism, and the ability to estimate the solution pointwise without a global solve.

While the WoS method shares conceptual similarities with our WoB approach, WoS operates in a bottom-up fashion, recursively applying a local relationship within spheres contained in the domain. In contrast, the WoB method adopts a top-down strategy: it first derives a BIE defined globally over the boundary and then employs Monte Carlo sampling to solve it, similarly to how Monte Carlo rendering operates.

Tangentially, while WoS and WoB are designed for volumetric PDEs, this thesis also explores the generalization of WoS to surface PDEs, as discussed in Chapter 6. Consequently, the methods we summarize here serve as the basis of PWoS.

In the following, we summarize the WoS method and review recent advancements in WoS-type techniques that were developed concurrently with this thesis work. Further details can be found in the PhD thesis by Sawhney [2024].

2.3.1 Solution Estimate for the Poisson Equation

WoS solves volumetric PDEs such as the Poisson equation over a bounded domain Ω in $\mathbb{R}^d$, and the problem we consider here is an interior Dirichlet problem for the Poisson equation. To estimate the solution at point $\mathbf{x} \in \Omega$, we first consider a d-ball $B_R(\mathbf{x})$, centered at $\mathbf{x}$ with radius R, fully contained within the domain. We can easily compute the radius of such a ball via a closest point query against the boundary geometry $\partial\Omega$. The integral equation

$$u(\mathbf{x}) = \frac{1}{|\partial B_R(\mathbf{x})|} \int_{\partial B_R(\mathbf{x})} u(\mathbf{y}) \, \mathrm{d}\mathbf{y} + \int_{B_R(\mathbf{x})} f(\mathbf{z}) G_R(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z} \tag{2.17}$$

holds for the Poisson equation $\nabla^2 u = f$ in general, where $|\partial B_R(\mathbf{x})|$ denotes the surface area of the sphere that bounds the ball $B_R(\mathbf{x})$ and G_R denotes the Green's function for the Poisson equation on $B_R(\mathbf{x})$ [Sawhney and Crane 2020]. Note that this differs from the fundamental solution discussed in Section 2.2.2. A fundamental solution refers specifically to a Green's function defined in an infinite domain, whereas the Green's function in this

context is the one within the ball. On the right-hand side of Eq. 2.17, the first term is a boundary integral over the $(d-1)$-sphere, and the second term is a volume integral over the d-ball. If we perform Monte Carlo integration of the first term by uniformly sampling a point on the sphere and of the second term by sampling N_V points $\mathbf{z}_i$ inside the ball with PDF $p(\mathbf{z}_i)$, we get the recursive relationship used in WoS:

$$\widehat{u}(\mathbf{x}) = \widehat{u}(\mathbf{y}) + \frac{1}{N_V} \sum_{i=1}^{N_V} \frac{G_R(\mathbf{x}, \mathbf{z}_i) f(\mathbf{z}_i)}{p(\mathbf{z}_i)}, \tag{2.18}$$

where the hat notation indicates that a variable is a Monte Carlo estimate. The first term on the right-hand side is also a Monte Carlo estimate because the solution $u(\mathbf{y})$ is unknown at point $\mathbf{y}$ in general. At each recursion step, WoS applies this recursive relationship to the largest ball inside the domain bounded by Dirichlet boundaries. It terminates the recursion when the sample point $\mathbf{x}$ during the recursion falls within a small distance ϵ of the domain boundary, by using the known solution at the closest boundary point $\overline{\mathbf{x}}$ as the solution estimate: $\widehat{u}(\mathbf{x}) \coloneqq u(\overline{\mathbf{x}})$. WoS thereby estimates the solution at each evaluation point independently, offering intrinsic parallelism. The geometric flexibility and robustness of WoS also follow from the algorithmic design that uses only closest point queries.

2.3.2 Solution Estimate for the Screened Poisson Equation

There exists a generalization of WoS to the screened Poisson equation $\nabla^2 u - \sigma u = f$, where σ is a positive constant [Elepov and Mikhailov 1969; Sawhney and Crane 2020]. The modified integral equation for the screened Poisson equation is

$$u(\mathbf{x}) = \frac{c_R^\sigma}{|\partial B_R(\mathbf{x})|} \int_{\partial B_R(\mathbf{x})} u(\mathbf{y}) \, \mathrm{d}\mathbf{y} + \int_{B_R(\mathbf{x})} f(\mathbf{z}) G_R^\sigma(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z}, \tag{2.19}$$

where G_R^σ is the Green's function for the screened Poisson equation on $B_R(\mathbf{x})$ and c_R^σ is a positive number smaller than 1.

To evaluate the first term, instead of multiplying the contribution from the recursion by c_R^σ, we can use a Russian Roulette strategy: we terminate the path early with probability $1 - c_R^\sigma$ with zero contribution and otherwise we use the estimate of the solution without multiplying by c_R^σ. This scheme allows us to terminate the WoS path early without waiting for it to reach the boundary and without introducing additional bias for the screened Poisson equation. It also makes it possible to apply WoS to problems in an infinite domain. We can even use this estimator for the screened Poisson equation as a Tikhonov regularization

for the Poisson equation without boundaries: solving the screened Poisson equation with small σ yields an approximate solution to the Poisson equation. Similar regularization ideas appear in multiple contexts [Sabelfeld and Simonov 1994, Section 6.3; Sawhney and Miller et al. 2023].

2.3.3 Gradient Estiamte

With WoS, we can estimate the spatial gradient of the solution as well. The gradient estimator replaces the integral equation for the first step of the recursion with

$$\nabla u(\mathbf{x}) = \frac{1}{|B_R(\mathbf{x})|} \int_{\partial B_R(\mathbf{x})} u(\mathbf{y}) \mathbf{n}(\mathbf{y}) \, \mathrm{d}\mathbf{y} + \int_{B_R(\mathbf{x})} f(\mathbf{z}) \nabla_{\mathbf{x}} G_R(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z}, \tag{2.20}$$

where $|B_R(\mathbf{x})|$ is the volume of the ball and $\mathbf{n}(\mathbf{y})$ is the outward unit normal of the ball at $\mathbf{y}$. For the screened Poisson equation, we multiply the first term by c_R^σ and replace ∇G_R in the second term with ∇G_R^σ to get a similar integral equation. To evaluate the integrals, we can uniformly sample a point on the sphere for the first term, and generate the samples with, e.g., $p(\mathbf{z}_i) \propto 1/\|\mathbf{x} - \mathbf{z}_i\|_2^2$ in 3D, for the second term. The WoB algorithm we present in this thesis similarly allows gradient estimation by modifying the first step of the walk.

2.3.4 Neumann and Robin boundaries — Walk on Stars

The WoS method was originally proposed for solving Dirichlet problems [Muller 1956]. While some early extensions enabled WoS to handle Neumann and Robin boundary conditions [Ermakov and Sipin 2009; Mascagni and Simonov 2004; Simonov 2008], these approaches often suffered from limited efficiency. To address this issue, concurrently with the work in this thesis, Sawhney and Miller et al. [2023, 2024b] introduced a different generalization of the WoS method, which they termed the *Walk on Stars* (WoSt) method. In this thesis, we regard WoSt as a member of the broader WoS family, and denote this class of methods as WoS(t).

The intuition behind the design of WoSt is that Neumann boundary conditions should *reflect* random walks, in contrast to Dirichlet boundaries, which *absorb* them. Robin boundaries, in this framework, act as *partially absorbing* boundaries. WoSt replaces the ball-shaped domains used in the classic Walk on Spheres (WoS) method with *star-shaped* domains, enabling a recursive algorithm that naturally handles mixed boundary conditions.

Consider, for example, a Laplace problem defined on a domain Ω with a mixture of Dirichlet and homogeneous (or zero) Neumann boundary conditions. Around a given

evaluation point $\mathbf{x}$, we define a local star-shaped domain $\mathrm{St}(\mathbf{x})$ as the intersection of Ω with a ball of radius R centered at $\mathbf{x}$: $\mathrm{St}(\mathbf{x}) = \Omega \cap B_R(\mathbf{x})$. The radius R is chosen to be the largest possible such that (i) the nearest Dirichlet boundary point to $\mathbf{x}$ lies outside the ball, and (ii) any ray cast from $\mathbf{x}$ intersects at most one Neumann boundary point. Given this star shape, we sample a random direction uniformly on the unit sphere and cast a ray from $\mathbf{x}$ and take the intersection point as the next point in the recursive walk algorithm. If this point lies within a distance ϵ of a Dirichlet boundary, we treat it as an absorption event and use the boundary value as the contribution of that path. Otherwise, the process repeats. Crucially, the algorithm allows sample points to land on Neumann boundaries without terminating, thereby reflecting the walk in accordance with the Neumann condition.

This construction generalizes to more complex settings, including problems with Robin boundaries (where walks are partially absorbed), Poisson problems with nonzero source terms, and screened Poisson problems.

In the design of this algorithm, several geometric queries must be implemented, including the closest point query, the closest silhouette point query, and the ray intersection query. Sawhney and Miller et al. [2023, 2024b] proposed an efficient implementation of these queries for domains with polygonal boundaries in 2D and triangle mesh boundaries in 3D. However, it remains unclear whether these queries can be implemented efficiently with other boundary geometry representations. In contrast, WoB allows for more flexible sampling strategies, e.g., requiring only ray intersection and uniform surface-area sampling queries, which are supported across many geometry representations.

Another challenge of WoS(t) arises when dealing with exterior (or unbounded) domains. In such cases, random walks may fail to terminate naturally and escape to infinity, necessitating special treatment. Sawhney and Crane [2020] proposed a stochastic path termination scheme based on Russian roulette; however, this approach can lead to significantly increased variance. An alternative strategy is to apply the Kelvin transform, which maps the unbounded domain into a bounded one, ensuring that all paths eventually terminate [Nabizadeh et al. 2021]. Yet another method introduces an artificial exit sphere surrounding the evaluation point; walks that reach this sphere are either treated as having reached infinity or are remapped to continue from a point on the sphere [Zhou et al. 1994]. Despite their effectiveness, both approaches require substantially different treatments for interior and exterior problems. In contrast, WoB supports both interior and exterior domains within a single, unified algorithmic framework.

2.3.5 Other Extensions

Beyond the extensions of WoS discussed so far, Monte Carlo approaches for non-rendering tasks are gaining increasing attention in computer graphics and related fields, with several developments occurring concurrently with this thesis.

Notable efforts that broaden the applicability of WoS and related techniques include the following: Sawhney and Seyb et al. [2022] extended WoS to support problems with spatially varying coefficients. Bati et al. [2023] studied a related solver for infrared rendering problems. De Lambilly et al. [2023] applied WoS for real-time heat simulation. De Goes and Desbrun [2024] proposed an efficient use of WoS for cage-based deformation. Tian and Günther [2025] employed WoS to visualize two-dimensional flow fields. Miller et al. [2025] developed a solver for microparticle geometries modeled as participating media. Miller et al. [2024a]; Yilmazer et al. [2024]; Yu et al. [2024] proposed using WoS(t) to solve for the gradient of the solution with respect to some scene parameters, and applied them to optimization problems.

In parallel, there has been substantial work aimed at improving the efficiency of WoS. Qi et al. [2022] introduced a reverse WoS method, where walks originate from boundary or source points, akin to the global WoS strategy proposed by Sabelfeld and Kireeva [2020]. Miller and Sawhney et al. [2023] proposed a boundary value caching strategy, where they store required values on the boundary of the entire domain or a subdomain to enable the reuse of information across query points. Bakbouk and Peers [2023] and Czekanski et al. [2025] introduced variance reduction techniques that exploit spatial relationships among nearby queries. Himmler and Günther [2025] proposed to use non-spherical shapes to construct an algorithm similar to WoS to remove the epsilon-shell bias from the solver in 2D. Additionally, several works incorporate neural networks to either cache the solution for future queries or enhance the efficiency of WoS [Belhe et al. 2023; Huang et al. 2025; Li and Yang et al. 2023, 2024; Nam et al. 2024; Song et al. 2025].

While some of these approaches are deeply coupled with the core formulation of WoS, many are more broadly applicable. In particular, any method compatible with a black-box, pointwise Monte Carlo PDE solver can be used with WoB as well. Furthermore, techniques such as reverse walks and boundary value caching naturally arise in WoB due to its conceptual similarity to Monte Carlo rendering. We demonstrate these ideas in the context of WoB in Chapter 3.

As for PWoS, most methods discussed here are tailored for volumetric PDEs and thus are not directly applicable. However, we show in Chapter 6 how to adapt the volume mean value filtering strategy of Bakbouk and Peers [2023] to the surface setting.

Chapter 3

Monte Carlo PDE Solvers for General Boundary Value Problems

Building on the boundary integral equation (BIE) formulation of partial differential equations (PDEs) introduced in Chapter 2, we present the *Walk on Boundary* (WoB) method for solving boundary value problems. WoB is a grid-free Monte Carlo solver for *general boundary value problems*, capable of handling a wide range of boundary conditions—Dirichlet, Neumann, Robin, and mixed—in a unified, top-down framework. This stands in contrast to the previously proposed bottom-up Walk on Spheres (WoS) approach [Muller 1956; Sawhney and Crane 2020], which, in its basic form, was limited to interior Dirichlet problems. WoB shares with Monte Carlo ray tracing the core idea of formulating problems as BIEs over the domain boundary. This analogy enables the direct application of advanced variance reduction techniques from rendering to PDE solving, and we demonstrate several numerical results using such techniques. We begin by introducing WoB in the context of the Laplace equation, followed by simple extensions to support the Poisson equation. Chapter 4 will further generalize WoB to other classes of PDEs, and Chapter 5 will show an application to an even more complex problem.

3.1 Overview

Let us examine instances of WoS and WoB estimators to highlight the differences. Consider a boundary value problem in which a function $\overline{u}_D(\mathbf{x})$ is specified along the boundary of a convex domain D, for simplicity. We will relax this assumption of convexity in Section 3.2.

We seek a function $u(\mathbf{x})$ that conforms to $\overline{u}_D(\mathbf{x})$ on the boundary while satisfying the Laplace equation, $\nabla^2 u(\mathbf{x}) = 0$, inside the domain.

The WoS estimator in this case is defined as the mean of sample contributions $\hat{u}(\mathbf{x}_0)$ defined recursively as

$$\hat{u}(\mathbf{x}_i) = \begin{cases} \overline{u}_D(\mathbf{x}_i) & \mathbf{x}_i \in \partial D_\epsilon, \\ \hat{u}(\mathbf{x}_{i+1}) & \text{otherwise.} \end{cases} \tag{3.1}$$

Starting from an evaluation point $\mathbf{x}_0$ within the domain, each subsequent $\mathbf{x}_{i+1}$ is generated by sampling a point on the largest ball centered at $\mathbf{x}_i$ that fits within the domain. The set ∂D_ϵ is a shell-like region that lies within a small distance of the boundary. WoS thus returns $\overline{u}_D(\mathbf{x}_i)$ when $\mathbf{x}_i$ is only *approximately* on the boundary; indeed, none of the generated points are exactly on the boundary.

The WoB estimator with path length M we introduce is defined as

$$\hat{u}(\mathbf{x}_i) = \begin{cases} \overline{u}_D(\mathbf{x}_{i+1}) & i = M-1, \\ 2\overline{u}_D(\mathbf{x}_{i+1}) - \hat{u}(\mathbf{x}_{i+1}) & \text{otherwise.} \end{cases} \tag{3.2}$$

Like WoS, WoB forms a sequence of $\mathbf{x}_i$ starting from the evaluation point $\mathbf{x}_0$. Each $\mathbf{x}_{i+1}$, however, is generated by tracing a random ray from $\mathbf{x}_k$ and finding the intersection with the boundary. Unlike WoS, all the points (after $\mathbf{x}_0$) are *exactly* on the boundary, and thus $\overline{u}_D(\mathbf{x}_{i+1})$ is well-defined, without approximation via the ϵ-shell ∂D_ϵ. As in Monte Carlo ray tracing, the recursion terminates after some number of steps M and does not depend at all on the ϵ-shell ∂D_ϵ. Algorithm 1 summarizes this method.

Just like Monte Carlo ray tracing, in both WoS and WoB, one would run the estimator multiple times and take its average as a final estimation of $u(\mathbf{x}_0)$. However, because WoB is also based on sampling and tracing rays recursively, the analogy with Monte Carlo ray tracing is much stronger, in sharp contrast to the random walk on spheres process used by WoS. One of our contributions is to demonstrate WoB's strong mathematical and algorithmic similarity to Monte Carlo ray tracing.

The ray tracing approach to diffusion curves [Bowers et al. 2011] also solves a boundary integral and estimates the solution by sampling points on the boundary using ray intersections, similarly to WoB. While this approach provides a *visually* faithful approximation for diffusion curves [Orzan et al. 2008], it does not actually solve the Laplace equation. By contrast, the WoB formulation does solve the specified PDE (i.e., the Laplace equation) and can handle more general problems, such as Neumann problems.

Algorithm 1: Interior, convex-domain, and Dirichlet WoB

Input: domain Ω, evaluation point $\mathbf{x}_0 \in \Omega$, path length M, sample count N

Function `EstimateSolution`(Ω, $\mathbf{x}_0$, M, N)**:**

- $\hat{u}_{\text{sum}} \leftarrow 0$
- **for** $n \leftarrow 1$ *to* N **do**
  - $\hat{u} \leftarrow$ `RecursiveEstimate` (Ω, $\mathbf{x}_0$, M, 0)
  - $\hat{u}_{\text{sum}} \leftarrow \hat{u}_{\text{sum}} + \hat{u}$
- **end**
- **return** $\hat{u}_{\text{sum}}/N$

Function `RecursiveEstimate`(Ω, $\mathbf{x}_i$, M, *depth*)**:**

- $\mathbf{x}_{i+1} \leftarrow$ `RayIntersectionSampling`(Ω, $\mathbf{x}_i$)
- **if** *depth* $= M - 1$ **then**
  - **return** $\overline{u}_D(\mathbf{x}_{i+1})$
- **else**
  - $\hat{u}_{i+1} \leftarrow$ `RecursiveEstimate`(Ω, $\mathbf{x}_{i+1}$, M, `depth` $+ 1$)
  - **return** $2\overline{u}_D(\mathbf{x}_{i+1}) - \hat{u}_{i+1}$
- **end**

Function `RayIntersectionSampling`(Ω, $\mathbf{x}$)**:**

- $\mathbf{d} \leftarrow$ `UniformRayDirectionSampling`(Ω, $\mathbf{x}$)
- **return** `GetIntersectionPoint`(Ω, `Ray`($\mathbf{x}$, $\mathbf{d}$))

3.2 Walk on Boundary Method

WoB is a stochastic estimator for certain classes of second-order PDEs based on BIEs. It uses a sequence of stochastically chosen sample points on the boundary, hence the name "Walk on Boundary". Sabelfeld [1982] first proposed WoB for the Lamé equation for linear elasticity. Subsequent books by Sabelfeld [1991] and Sabelfeld and Simonov [1994] generalized the method and established theoretical foundations for Dirichlet, Neumann, and Robin problems for the Poisson equation using the indirect BIE formulation, along with extensions to a few other equations, including screened Poisson, linear elastostatics, and diffusion problems. Karaivanova et al. [2004] considered the applicability of quasi-Monte Carlo methods, and Sabelfeld [2012] studied how WoB can be combined with the method of fundamental solutions to improve its efficiency.

None of this prior work has considered the application of WoB ideas to direct BIE formulations or mixed boundary problems; these extensions are our contributions. We have also identified that using double layer potential indirect BIEs for Dirichlet boundary problems, single layer potential indirect BIEs or direct BIEs for Neumann boundary problems, and single layer potential indirect BIEs for Robin and mixed boundary problems will result in practical solvers. We summarize the equations for formulations for all of our estimators discussed in this paper in Table 3.1.

3.2.1 Dirichlet Problems with Double Layer BIE

For Dirichlet problems, reordering terms in Eq. 2.14 and substituting in the boundary condition $u = \overline{u}_D$ gives

$$\nu(\mathbf{x}) = \int_\Gamma 2\frac{\partial G}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}, \mathbf{y})\nu(\mathbf{y})\,\mathrm{d}\mathbf{y} + 2\overline{u}_D(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \Gamma, \tag{3.3}$$

assuming that $\mathbf{x}$ lies on a smooth boundary ($c = 1/2$). Since $\overline{u}_D(\mathbf{x})$ is a known quantity, Eq. 3.3 is a Fredholm equation of the second kind for ν, as the rendering equation is also commonly understood to be. Section 3.4.1 elaborates on this point. We can thus apply a recursive estimate for $\nu(\mathbf{x})$ similarly to what is done for the rendering equation in light transport simulation [Pharr et al. 2018].

Monte Carlo Estimation

Let us consider estimating $\nu(\mathbf{x}_1)$ where $\mathbf{x}_1 \in \Gamma$ based on Monte Carlo integration. We can estimate the integral in Eq. 3.3 via Monte Carlo integration by first sampling a point

Table 3.1: List of equations for WoB estimators. The highlighted equations are the second kind Fredholm equations (or the modified first-kind equation for mixed boundary problems at Dirichlet boundaries) we use to get the unknown direct or indirect quantities on the boundary. It can be combined with other equations to find the unknown quantities of interest in the interior (or exterior) or on the boundary. It is assumed that the left hand side unknowns are functions of $\mathbf{x}$, and the integrals are taken over boundary points $\mathbf{y}$. The explicit dependencies on variables are omitted for brevity when it is not confusing. For interior problems, $\phi = 1$, and for exterior problems, $\phi = -1$.

Problem	Formulation	Quantity to Estimate	Evaluation Point $\mathbf{x}$	Equation
Dirichlet	indirect BIE double layer potential	solution	interior/exterior	$u = -\int \frac{\partial G}{\partial \mathbf{n_y}} \nu \, dA$
			boundary	$u = \overline{u}_D$ (given)
		normal derivative	boundary	[Sabelfeld and Simonov 1994, Section 3]
		gradient	interior/exterior	$\frac{\partial u}{\partial \mathbf{x}_k} = -\int \frac{\partial^2 G}{\partial \mathbf{x}_k \partial \mathbf{n_y}} \nu \, dA$
		source density	**boundary**	$\nu = 2\phi \int \frac{\partial G}{\partial \mathbf{n_y}} \nu \, dA + 2\phi \overline{u}_D$
Neumann	direct BIE	**solution**	interior/exterior	$u = -\phi \int \frac{\partial G}{\partial \mathbf{n_y}} u \, dA + \phi \int G \overline{q}_N \, dA$
			boundary	$u = -2\phi \int \frac{\partial G}{\partial \mathbf{n_y}} u \, dA + 2\phi \int G \overline{q}_N \, dA$
		normal derivative	boundary	$\frac{\partial u}{\partial \mathbf{n}} = \overline{q}_N$ (given)
		gradient	interior/exterior	$\frac{\partial u}{\partial \mathbf{x}_k} = -\phi \int \frac{\partial^2 G}{\partial \mathbf{x}_k \partial \mathbf{n_y}} u \, dA + \phi \int \frac{\partial G}{\partial \mathbf{x}_k} \overline{q}_N \, dA$
Mixed or degenerate problem	indirect BIE single layer potential	solution	interior/exterior/boundary	$u = \int G\mu \, dA$ (or given)
		normal derivative	boundary	$\frac{\partial u}{\partial \mathbf{n}} = \frac{1}{2}\phi\mu + \int \frac{\partial G}{\partial \mathbf{n_x}} \mu \, dA$ (or given)
		gradient	interior/exterior	$\frac{\partial u}{\partial \mathbf{x}_k} = \int \frac{\partial G}{\partial \mathbf{x}_k} \mu \, dA$
		source density	**Dirichlet boundary**	$\mu = \mu - k \int G\mu \, dA + k\overline{u}_D$
			Neumann boundary	$\mu = -2\phi \int \frac{\partial G}{\partial \mathbf{n_x}} \mu \, dA + 2\phi \overline{q}_N$
			Robin boundary	$\mu = -2\phi \int \left(\frac{\partial G}{\partial \mathbf{n_x}} + \overline{\alpha}(\mathbf{x}) G \right) \mu \, dA + 2\phi \overline{g}_R$

$\mathbf{x}_2 \in \Gamma$ with a probability density function (PDF) $p(\mathbf{x}_2|\mathbf{x}_1)$ (e.g., tracing a random ray from $\mathbf{x}_1$ to $\mathbf{x}_2$). A sample $\hat{\nu}(\mathbf{x}_1)$ to estimate $\nu(\mathbf{x}_1)$ can be written as

$$\hat{\nu}(\mathbf{x}_1) \coloneqq \frac{2\frac{\partial G}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}_1, \mathbf{x}_2)}{p(\mathbf{x}_2|\mathbf{x}_1)}\nu(\mathbf{x}_2) + 2\overline{u}_D(\mathbf{x}_1) \quad \text{for} \quad \mathbf{x}_1 \in \Gamma. \tag{3.4}$$

Because $\nu(\mathbf{x}_2)$ is unknown, we again use a Monte Carlo estimate $\hat{\nu}(\mathbf{x}_2)$ in the equation above. Thus a recursive definition for the i-th step is:

$$\hat{\nu}(\mathbf{x}_i) \coloneqq \frac{2\frac{\partial G}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}_i, \mathbf{x}_{i+1})}{p(\mathbf{x}_{i+1}|\mathbf{x}_i)}\hat{\nu}(\mathbf{x}_{i+1}) + 2\overline{u}_D(\mathbf{x}_i) \quad \text{for} \quad \mathbf{x}_i \in \Gamma. \tag{3.5}$$

Just like Monte Carlo ray tracing, we perform the recursive estimate of $\nu(\mathbf{x})$ up to a certain recursion depth M, forming a path of vertices on the boundary with length M. We can use $\hat{\nu}(\mathbf{x}_1)$ to construct a Monte Carlo estimate for the solution u at an interior point $\mathbf{x}_0$. Applying another Monte Carlo integration with a PDF $p(\mathbf{x}_1|\mathbf{x}_0)$ and $\hat{\nu}(\mathbf{x}_1)$ to Eq. 2.13 gives

$$\hat{u}(\mathbf{x}_0) \coloneqq -\frac{\frac{\partial G}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}_0, \mathbf{x}_1)}{p(\mathbf{x}_1|\mathbf{x}_0)}\hat{\nu}(\mathbf{x}_1) \quad \text{for} \quad \mathbf{x}_0 \in \Omega. \tag{3.6}$$

Therefore, the Monte Carlo estimate for $u(\mathbf{x}_0)$ in the domain interior is $u(\mathbf{x}_0) \approx \frac{1}{N}\sum_{n=1}^{N}\hat{u}(\mathbf{x}_0)$. One can think of the PDF $p(\mathbf{x}_{i+1}|\mathbf{x}_i)$ as the PDF of sampling a ray from $\mathbf{x}_i$ toward $\mathbf{x}_{i+1}$, the term $2\frac{\partial G}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}_i, \mathbf{x}_{i+1})$ as the geometry term times BRDF term (i.e., the integrand) of the rendering equation, and the term $2\overline{u}_D(\mathbf{x}_i)$ as the emission term in the rendering equation. Implementation of WoB on top of ray tracing systems is thus straightforward.

Path Truncation

One difference between the rendering equation and the above BIE is that, for the rendering equation, as the path length increases, the contribution coming from each recursion becomes smaller and smaller due to the nature of light transport. However, the integral kernel $2\frac{\partial G}{\partial \mathbf{n}_\mathbf{y}}$ above will not "attenuate" its contribution per recursion, but will rather maintain it. It thus appears that it never converges, just like having reflectance equal to one everywhere does not converge in light transport. This intuition contradicts the fact that solutions usually uniquely exist for Dirichlet problems, and it is incorrect.

A solution to this issue is surprisingly simple. We just need to multiply the contribution $2\overline{u}_D(\mathbf{x}_i)$ coming from the last recursion step by a factor of $1/2$ before it terminates: $\hat{\nu}(\mathbf{x}_M) \coloneqq \overline{u}_D(\mathbf{x}_M)$. While this strategy deceivingly looks the same as just truncating a path

while reducing the contribution from the last "bounce", its derivation is more involved than that. Below, we briefly summarize the rough reasoning behind this strategy.

Let us first define an integral operator H when applied to a function f defined over the boundary as

$$(\mathrm{H}\, f)(\mathbf{x}) = \int_\Gamma 2\frac{\partial G}{\partial \mathbf{n_y}}(\mathbf{x}, \mathbf{y}) f(\mathbf{y})\, \mathrm{d}\mathbf{y}. \tag{3.7}$$

Then, Eq. 3.3 can be rewritten as

$$\nu = \mathrm{H}\,\nu + 2\overline{u}_D, \tag{3.8}$$

where we dropped the variable dependence for brevity. Using the identity operator I, we can write the expression above as

$$(\mathrm{I} - \mathrm{H})\nu = 2\overline{u}_D. \tag{3.9}$$

One can use the Neumann series expansion to solve for ν as

$$\nu = (\mathrm{I} - \mathrm{H})^{-1}\, 2\overline{u}_D = (\mathrm{I} + \mathrm{H} + \mathrm{H}^2 + \cdots)\, 2\overline{u}_D, \tag{3.10}$$

where the operator H^i for any positive integer i is defined by

$$\mathrm{H}^i\, f = \mathrm{H}^{i-1}(\mathrm{H}\, f). \tag{3.11}$$

The same approach is used for building a recursive Monte Carlo estimator for the rendering equation, where the operator is defined by BRDF and the geometry term instead [Pharr et al. 2018].

As noted earlier, the key difference from the rendering equation is that $(\mathrm{H}^i)2\overline{u}_D(\mathbf{x})$ does not approach zero as i increases. Simply truncating this series at M thus introduces non-negligible truncation error. We instead transform the series as

$$\begin{aligned}\left(\frac{1}{2}(\mathrm{I} + \cdots + \mathrm{H}^{i-1} + \cdots) + \frac{1}{2}(\mathrm{I} + \cdots + \mathrm{H}^{i} + \cdots)\right) 2\overline{u}_D(\mathbf{x}) \\ = \frac{1}{2}(\mathrm{I} + (\mathrm{I} + \mathrm{H}) + \cdots + (\mathrm{H}^{i-1} + \mathrm{H}^i) + \cdots) 2\overline{u}_D(\mathbf{x}).\end{aligned} \tag{3.12}$$

Just like multiple bounces in light transport, the average of the integrals $(1/2)(\mathrm{H}^i + \mathrm{H}^{i+1})2\overline{u}_D(\mathbf{x})$ now converges to zero as $i \to \infty$ due to the alternating sign in the series because of the

negative factor included in the operator H, so it is safe to truncate this modified series at $i = M$:

$$\begin{aligned} f(\mathbf{x}) &\approx \frac{1}{2}\left(\mathrm{I} + (\mathrm{I} + \mathrm{H}) + \cdots + (\mathrm{H}^{M-1} + \mathrm{H}^{M})\right) 2\overline{u}_D(\mathbf{x}) \\ &= \left(\mathrm{I} + \cdots + \mathrm{H}^{M-1} + \frac{1}{2}\,\mathrm{H}^{M}\right) 2\overline{u}_D(\mathbf{x}) \end{aligned} \tag{3.13}$$

Therefore, the term for the last point $\mathbf{x}_M$ should now be multiplied by $1/2$, when compared to just truncating the original series at the M-th term. For such fixed-length truncation, the resulting estimator is biased, just like Monte Carlo rendering with a finite path length. One could potentially apply the Russian roulette technique to truncate the path without introducing bias, but this requires a careful investigation of its impact on the modified Neumann series. We leave this as future work.

Formally, this transformation of a Neumann series can be mathematically interpreted as an analytic continuation of the series, and many other transformations are possible [Sabelfeld 1991; Sabelfeld and Simonov 1994]. Sabelfeld [1991] numerically compares some transformations, and our initial experiments also suggest that series acceleration, such as the van Wijngaarden transformation [van Wijngaarden 1953], can reduce the error of the estimator based on Neumann series. While the optimal path truncation strategy and the weights assigned to contributions from different path lengths would vary depending on the specific problem, and the discussion here does not strictly apply beyond interior Dirichlet problems, all results presented in this work use the modified Neumann series with a factor of $1/2$ applied to the final term.

Derivative and Boundary Value Estimators

With WoB, in addition to the solution within the domain, we can easily estimate the gradient inside the domain, the solution on the boundary, and the normal derivative on the boundary, just by replacing the first step of our recursive Monte Carlo solution estimators. For example, we can apply a Monte Carlo estimator based on the equation for the interior gradient, $\frac{\partial u}{\partial \mathbf{x}_k} = -\int \frac{\partial^2 G}{\partial \mathbf{x}_k \partial \mathbf{n}_\mathbf{y}} \nu \, dA$, instead of Eq. 3.6, to get a gradient estimate in the case of the Dirichlet problem estimator with double layer potential formulation. Similar to WoS [Sawhney and Crane 2020], we can reuse the same paths to get samples for the solution and the gradient in the interior by changing the initial weight at almost no additional cost. We, however, observed increased noise near the boundary in the gradient estimates for this Dirichlet problem estimator. This additional noise is likely due to the presence of the hypersingular kernel $\frac{\partial^2 G}{\partial \mathbf{x}_k \partial \mathbf{n}_\mathbf{y}}$ in the computation, which has a very large variance when the

interior point $\mathbf{x}$ is placed very close to a sampled boundary point $\mathbf{y}$. To overcome this issue, we could separate the gradient into the normal and tangential components defined with respect to the nearest boundary point and carefully evaluate each term as described by Sabelfeld and Simonov [1994].

The estimates of the normal derivatives on the boundary with WoB use the equations we get by taking the limit of the integral equations for gradient estimation to the boundary. For this Dirichlet estimator, however, the normal derivative estimator derived this way involves a hypersingular integral, which has an infinite variance and cannot be used directly; we will need to transform it to another form for evaluation [Sabelfeld and Simonov 1994].

3.2.2 Neumann Problems with Direct BIE

The strength of WoB is that we can apply essentially the same approach of building a recursive Monte Carlo estimator to address other boundary problems than Dirichlet problems. While Sabelfeld and Simonov [1994] proposed using single layer potentials to solve Neumann problems, we propose another formulation based on direct BIEs since this formulation allows us to utilize a different estimator than the one with single layer potentials, as we will explain later. Our formulation uses Eq. 2.8 in combination with the Neumann boundary conditions $\overline{q}_N$ as

$$u(\mathbf{x}) = -\int_\Gamma 2\frac{\partial G}{\partial \mathbf{n_y}}(\mathbf{x},\mathbf{y})u(\mathbf{y})\,\mathrm{d}\mathbf{y} + \int_\Gamma 2G(\mathbf{x},\mathbf{y})\overline{q}_N(\mathbf{y})\,\mathrm{d}\mathbf{y}, \tag{3.14}$$

at a point $\mathbf{x}$ on the boundary; the second term can be estimated with another Monte Carlo estimator without recursion because $\overline{q}_N$ is a known boundary value. We have found that it usually suffices to sample one boundary point to estimate the integral of the second term per recursion, though it is possible to have more samples.

The rest is similar to the Dirichlet case described above — we expand u recursively. We can estimate the unknown $u(\mathbf{x}_i)$ as

$$\hat{u}(\mathbf{x}_i) \coloneqq -\frac{2\frac{\partial G}{\partial \mathbf{n_y}}(\mathbf{x}_i,\mathbf{x}_{i+1})}{p_1(\mathbf{x}_{i+1}|\mathbf{x}_i)}\hat{u}(\mathbf{x}_{i+1}) + \frac{2G(\mathbf{x}_i,\mathbf{x}'_{i+1})}{p_2(\mathbf{x}'_{i+1}|\mathbf{x}_i)}\overline{q}_N(\mathbf{x}'_{i+1}). \tag{3.15}$$

In general, the two PDFs p_1 and p_2 can differ, sampling two distinct points $\mathbf{x}_{i+1}$ and $\mathbf{x}'_{i+1}$ based on the current point $\mathbf{x}_i$. This estimator can be used to estimate the interior value based on Eq. 2.8 with

$$\hat{u}(\mathbf{x}_0) \coloneqq -\frac{\frac{\partial G}{\partial \mathbf{n_y}}(\mathbf{x}_0,\mathbf{x}_1)}{p_1(\mathbf{x}_1|\mathbf{x}_0)}\hat{u}(\mathbf{x}_1) + \frac{G(\mathbf{x}_0,\mathbf{x}'_1)}{p_2(\mathbf{x}'_1|\mathbf{x}_0)}\overline{q}_N(\mathbf{x}'_1), \tag{3.16}$$

where $\mathbf{x}_0$ is an interior point and $\mathbf{x}_1$ and $\mathbf{x}_1'$ are boundary points. Estimating the gradient is also possible, but with a high variance, similar to the case of the Dirichlet problem estimator.

3.2.3 Mixed Boundary Problems with Single Layer BIE

For mixed boundary and Robin problems, we adopt a single layer potential formulation (Eq. 2.10) where the boundary unknown we need to estimate is μ. This formulation was used for pure Dirichlet, Neumann, and Robin problems by Sabelfeld and Simonov [1994], but not for mixed boundary problems. This formulation leads to a Fredholm equation of the second kind for parts of the boundary where Neumann or Robin conditions are specified (see Table 3.1 for the equations). However, for parts of the boundary where Dirichlet boundary conditions are specified, this formulation would result in an equation in the form of a Fredholm equation of the *first* kind:

$$\overline{u}_D(\mathbf{x}) = \int_\Gamma G(\mathbf{x}, \mathbf{y})\mu(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Gamma_D. \tag{3.17}$$

Unlike the second kind equation, the unknown quantity μ appears only inside the integral. One cannot simply apply recursive Monte Carlo estimation for equations in the form of a first kind equation since there is no recursion relationship we can utilize. A common approach is to discretize and solve the corresponding matrix equation, which ruins the advantages of WoB over the conventional alternatives.

Inspired by a similar technique by Sabelfeld and Simonov [1994], we propose to transform Eq. 3.17 by multiplying by a nonzero constant k on both sides and adding $\mu(\mathbf{x})$ to both sides:

$$\mu(\mathbf{x}) = \mu(\mathbf{x}) - k\int_\Gamma G(\mathbf{x}, \mathbf{y})\mu(\mathbf{y})\,\mathrm{d}\mathbf{y} + k\overline{u}_D(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \Gamma. \tag{3.18}$$

This equation now has a structure similar to the second kind equation, with the additional $\mu(\mathbf{x})$ on the right-hand side. This equation can be estimated recursively, just like the second kind equation. We estimate the unknown quantity on the left-hand side by sampling the next point, and estimate the contribution by

$$\hat{\mu}(\mathbf{x}_i) \coloneqq \begin{cases} -\frac{1}{p_k} \cdot k\frac{G(\mathbf{x}_i, \mathbf{x}_{i+1})}{p(\mathbf{x}_{i+1}|\mathbf{x}_i)}\hat{\mu}(\mathbf{x}_{i+1}) + k\overline{u}_D(\mathbf{x}_i) & \text{with prob. } p_k \\ \frac{1}{1-p_k}\hat{\mu}(\mathbf{x}_{i+1}) + k\overline{u}_D(\mathbf{x}_i) & \text{with prob. } 1 - p_k. \end{cases} \tag{3.19}$$

We sample one of the first two terms in Eq. 3.18 based on the given probability p_k. In the second case, when we sample the term $\mu(\mathbf{x})$, we remain at the same point, i.e., $\mathbf{x}_{i+1} \coloneqq \mathbf{x}_i$,

in the next recursion step. With this method of handling the first kind equation, we can construct a recursive Monte Carlo estimator for μ for mixed boundary problems to recover the solution by another Monte Carlo integration of Eq. 2.10.

The choice of the multiplication constant k is critical in this estimator, and we picked a value by trial and error. If it is too small, the bias is higher given the same path length. If it is too large, the Neumann series diverges and the estimator fails as justified by [Sabelfeld and Simonov 1994] in the case of pure Dirichlet problems. Sabelfeld and Simonov [1994] also discuss the restriction on the mixture weight $\overline{\alpha}$ of Robin boundary problems. We found that deriving theoretical bounds in the case of mixed boundaries is extremely involved and is left for future work.

The gradient estimator with this formulation does not exhibit the additional noise as with the other formulations because this formulation uses an integral kernel with a lower order of singularity. For the normal derivative estimator, we get μ in two terms, in the integral and outside of the integral. We need to sample either one of the terms or run two paths for the estimates of μ for the two terms.

One could alternatively employ direct BIEs to get a similar estimator for mixed boundary problems. However, we would still encounter a first kind equation, and in practice, the resulting schemes are not much different from the indirect BIE-based method above.

3.2.4 Sampling Strategies

WoB can use various strategies to sample paths, as in Monte Carlo ray tracing, and a well-designed strategy can sharply reduce the variance of the estimator. Both WoB and Monte Carlo ray tracing have a vast design space of sampling strategies for different problems, and our proposed strategies in this paper are by no means exhaustive. We leave further exploration of different strategies as future work and briefly explain the strategies we implemented in the following.

Ray Intersection Sampling

The integral kernel $2\frac{\partial G}{\partial \mathbf{n_y}}$ is in fact proportional to the differential solid angle of $\mathbf{y}$ from another point $\mathbf{x}$. Similar to the fact that sampling a ray will cancel out the geometry term in rendering, we can use ray tracing from $\mathbf{x}$ to perfectly importance sample $2\frac{\partial G}{\partial \mathbf{n_y}}$ at any $\mathbf{y}$. The main difference from rendering is that we do not have the visibility term between $\mathbf{y}$ and $\mathbf{x}$. We thus have to sample a ray from a sphere, not a hemisphere, and

sample all the intersection points (not just the first hit) along the ray from $\mathbf{x}_i$ via "all-hits" ray intersection queries in general. Such all-hits queries are available and used in ray tracing [Gribble et al. 2014].

Due to our recursive formulation, using all hits would cause exponential branching of paths, which may not be ideal for GPU ray tracing [Parker et al. 2010]. We can instead pick one intersection out of the m such intersections at random, which results in multiplying the PDF by $1/m$, and the sample contribution is thus multiplied by m but with no branching in recursion. We used this approach in all of our results in this paper. Since this approach leads to an exponential increase in variance instead of exponential computation cost per sample, we do not claim that this strategy is always better than using all hits and branching exponentially.

Although selecting one intersection uniformly may seem counterintuitive, this strategy naturally samples distant points with smaller probabilities than closer points. This sampling strategy correctly reflects the structure of the underlying PDE, where even faraway points may have small influences on the solution.

For the special case when the domain is convex, we revert to the original hemispherical sampling strategy with closest-hit query and the PDF in this strategy simplifies to $p(\mathbf{y}|\mathbf{x}) = -2\frac{\partial G}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}, \mathbf{y})$ because there is only one hit for a direction within the hemisphere. The solution estimator for the Dirichlet problem with path length M then becomes $\hat{u}(\mathbf{x}_0) = 2[u_D(\mathbf{x}_1) - u_D(\mathbf{x}_2) + \cdots] + (-1)^M u_D(\mathbf{x}_M)$. This estimator is the one introduced in Section 3.1 (and Algorithm 1).

Another strategy is to use only the first hit point, but combine it with direct sampling of a point on the boundary (like sampling a point on light sources in rendering) via multiple importance sampling (MIS) [Veach and Guibas 1995]. While the first-hit-only strategy would not cover the entire integration domain, MIS ensures an unbiased estimator since direct sampling of a point will cover the entire domain. We did not employ this strategy since it often had higher variance in our experiments. However, similar to different strategies in Monte Carlo rendering, there might be certain scenarios where this particular strategy works better than the others. We suggest that readers explore different options to determine a suitable strategy for a given problem.

Backward Estimator

We have so far considered tracing a path by sampling a series of points starting from the evaluation point $\mathbf{x}_0$. We call this estimator a "backward" estimator, consistent with "backward tracing" in rendering, which traces a path of light in a backward manner from

the sensor (pixel) all the way to the light source. One can think of our evaluation point as a pixel and a boundary as a light source. Note that Qi et al. [2022] adopted the opposite definitions of backward and forward from those in rendering, so their forward estimator corresponds to our backward estimator.

Forward Estimator

Sabelfeld and Simonov [1994] proposed an *adjoint estimator* in WoB, which forms paths starting from a point on the boundary. We have found that it is analogous to light tracing in rendering, which traces a path starting from a point on the light source and can also be explained via the adjoint of the rendering equation [Christensen 2003]. We call it a "forward estimator" as in "forward tracing" in rendering. In the forward estimator, we will need to make an explicit connection between each point along the path and the evaluation point to compute the contribution of a path to the evaluation point. The forward estimator has less variance than the backward estimator in some of the formulations where the integral kernel is proportional to $\frac{\partial G}{\partial \mathbf{n_x}}$ as opposed to $\frac{\partial G}{\partial \mathbf{n_y}}$. In this case, we want to generate a ray from $\mathbf{y}$ to sample a point $\mathbf{x}$ proportional to its differential solid angle. For Neumann problems, the formulation with single layer potential [Sabelfeld and Simonov 1994] is easier to importance sample by a forward estimator, while our formulation with the direct BIE matches better with a backward estimator.

Other Strategies

Importance sampling only the integral kernel does not perfectly importance sample all the terms, which is also true in rendering. For example, the Dirichlet boundary value $\overline{u}_D$ will not be importance sampled that way. Because our WoB is based on a Fredholm equation of the second kind and we can use ray tracing, it is easy to apply more advanced sampling techniques used in rendering, such as MIS, resampled importance sampling (RIS) [Talbot et al. 2005], Markov chain Monte Carlo (MCMC) [Kelemen et al. 2002; Veach and Guibas 1997], or the zero variance theory [Křivánek and d'Eon 2014] to implement WoB. It contrasts with WoS where applications of these techniques are not necessarily straightforward (e.g., an MIS bidirectional estimator is not available in WoS [Qi et al. 2022]). We show preliminary results with bidirectional estimators with MIS, resampling via RIS, MCMC WoB, and path reuse as in the virtual point lights method [Keller 1997], but many other techniques can be made available to WoB. One interesting aspect of WoB that might lead to further development of sampling strategies is that samples can have positive or negative contributions, unlike (forward) rendering, where all contributions are non-negative.

We might be able to gain insights from differentiable rendering, where they similarly have both positive and negative contributions [Chang et al. 2023].

3.2.5 Generalization

Exterior Problems

WoB efficiently handles exterior problems in its basic form. Instead of the domain Ω, we can solve the Laplace equation in $\mathbb{R}^2\backslash\Omega$ or $\mathbb{R}^3\backslash\Omega$ for exterior problems. We define the normals to remain oriented outward from the interior domain Ω and replace the definitions of the boundary values with those obtained by taking the limit from the exterior domain. Moreover, in addition to the boundary conditions, we require that the solution $u(\mathbf{x})$ approaches zero at infinity for exterior problems. The solvers for the exterior domain largely remain the same as for the interior domain, except for a few sign changes in the terms of the BIEs (see Table 3.1). Since WoB relies on neither ϵ-shell approximation nor closest point queries in WoS, its accuracy and performance for exterior domains remain the same as for interior problems. For example, WoB does not need Kelvin transformations [Nabizadeh et al. 2021].

Of the two Dirichlet problem estimators, the double layer potential formulation estimator requires additional attention when used for exterior problems. This formulation allows us to find solutions that decay according to $O(|\mathbf{x}|^{1-d})$ as $|\mathbf{x}| \to \infty$, where d is the dimension of the problem. It thus cannot handle more general cases where solutions decay as $O(|\mathbf{x}|^{2-d})$. Sabelfeld and Simonov [1994] explain how to generalize the double layer potential formulation for such cases. The basic idea is to reduce the problem to one without the slowly decaying component with some precomputation before applying WoB. We enabled this extension for the scenes in Figures 3.3 and 3.7 where we expect the decay rate $O(|\mathbf{x}|^{2-d})$, while we did not do so for the other examples for which we use analytical solutions with decay rate $O(|\mathbf{x}|^{1-d})$. The single layer potential formulation, on the other hand, supports solutions with a decay rate $O(|\mathbf{x}|^{2-d})$ without modification. The exterior problem estimators for Neumann, Robin, and mixed boundary problems based on the single layer potential formulation are also general enough to handle the general decay rate of harmonic functions.

Multiply-Connected Domain Problems

We mainly focus on simply connected domains, e.g., domains without holes inside, for simplicity, with the exception of the interior Neumann problem in Fig. 3.5. Multiply

connected domains require some additional considerations [Sabelfeld and Simonov 1994]. For Neumann problems with single layer potential, we can apply the same WoB estimators without modifications as long as the standard compatibility condition that the integral of the normal derivative evaluates to zero for each connected domain is satisfied. For Dirichlet (with double layer potential) and Robin problems (with single layer potential), the situation is more complicated. The applicability of WoB in its original form is guaranteed by assuming more artificial compatibility conditions in these cases, and we need to perform some precomputation to modify the problem similarly to the exterior Dirichlet problem estimator. We are yet to confirm the applicability and efficiency of these estimators by Sabelfeld and Simonov [1994] or to derive solvability conditions for Neumann problem estimators based on direct BIE formulation. For Dirichlet problems, we observed some successful applications of the single layer potential formulation with multiply-connected domains, but the restriction on the multiplicative constant k seems more strict, and it too requires further investigation.

Non-Zero Source Term

For the Poisson equation (i.e., $\bar{b}(\mathbf{x}) \neq 0$), we need to additionally sample an interior (or exterior) point to have a Monte Carlo estimate for the volume integral of V_0. We include this term when we retrieve the boundary values $v(\mathbf{x})$ or $\frac{\partial v}{\partial \mathbf{n}}(\mathbf{x})$ in the Laplace equation and when we compute the function V_0 in the relation $u(\mathbf{x}) = v(\mathbf{x}) - V_0(\mathbf{x})$. One strategy is to sample such a point uniformly within the domain, but a well-designed sampling strategy can reduce variance. For interior problems, one possible sampling strategy is to sample a point along a line that goes through the point $\mathbf{x}$. Another possibility is to sample the point depending on the distribution of the source term. Fig 3.9 shows examples for Dirichlet and Neumann problems by sampling interior points uniformly in terms of the area measure. We focus on the Laplace equation in the other results since the volume integral of V_0 can always be added trivially. Sawhney and Seyb et al. [2022] dismissed WoB as being formulated only for the Laplace equation, but as we show, WoB is certainly capable of handling the Poisson equation.

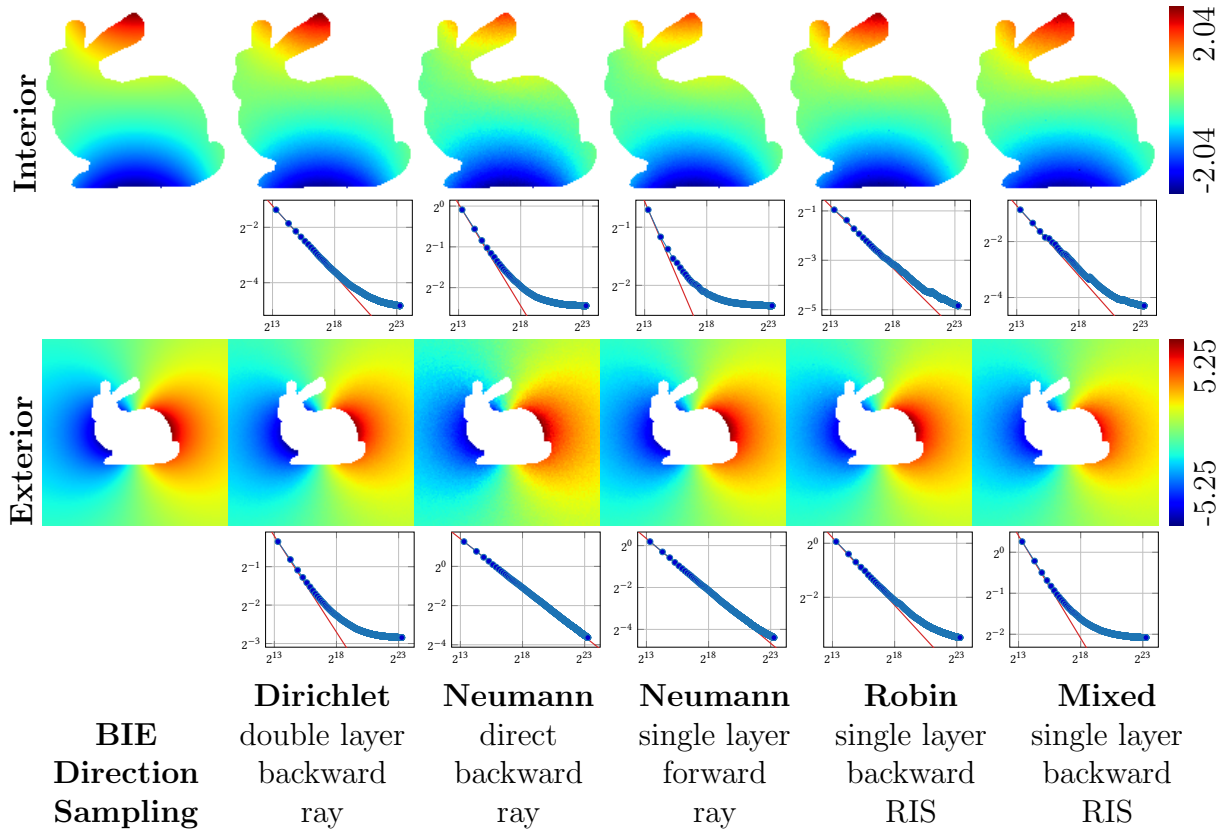


Figure 3.1: WoB applied to various problems. For the interior (top) and exterior (bottom) problems, we run WoB with path length $M = 4$ and $N = 10^7$ samples per evaluation point with the formulations and sampling techniques as labeled. The solution estimates are visualized with a color map. For each problem, we show the absolute root mean square error (vertical axis, varying scales) with respect to the number of samples (horizontal axis) measured against the reference analytical solution (left). The red lines show the $\mathcal{O}(1/\sqrt{N})$ decay rate for reference.

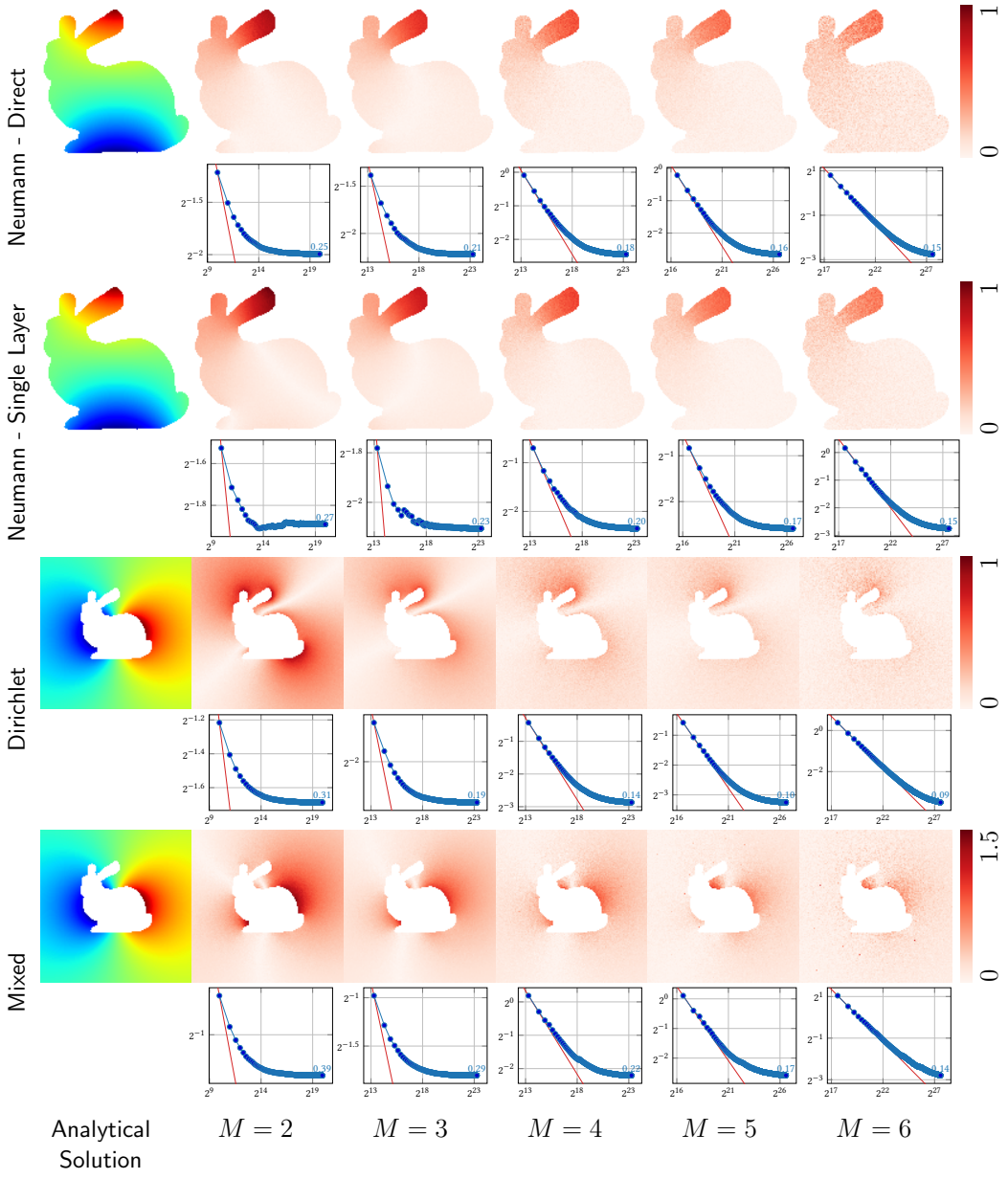


Figure 3.2: Path truncation error study. For the interior Neumann problem estimators and the exterior Dirichlet and mixed boundary problem estimators in Fig. 3.1, we show the absolute errors with different path lengths M. We show the absolute root mean square error (vertical axis) with respect to the number of samples (horizontal axis), and also show the remaining error in blue text. As path length increases, the bias decreases, but larger numbers of samples are needed to achieve convergence.

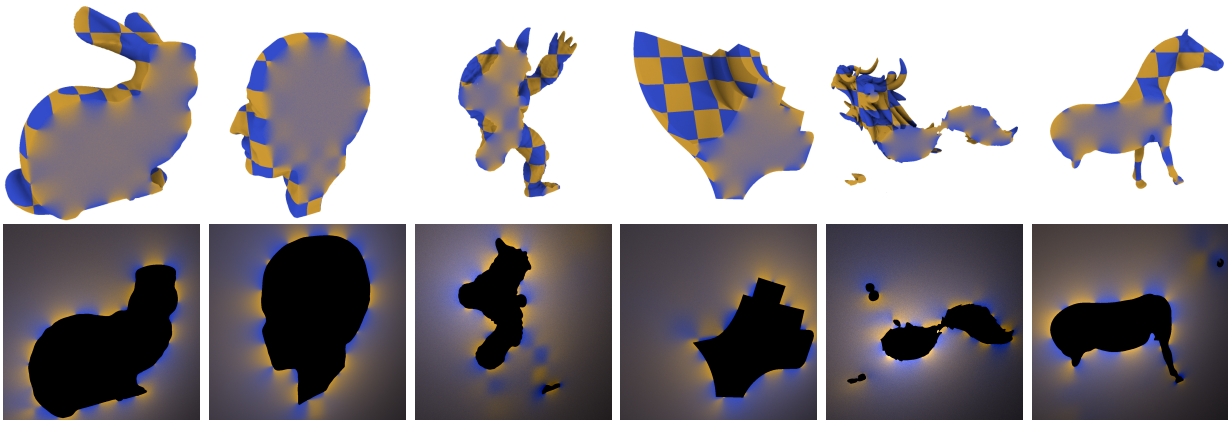


Figure 3.3: Results of a WoB solver implemented on top of a Monte Carlo ray tracing system. WoB's strong similarity to Monte Carlo ray tracing makes such an implementation easy to carry out. The images show the estimated solution for interior (top) and exterior (bottom) Dirichlet problems on a cutting plane. WoB can solve both problems efficiently with a unified Monte Carlo ray tracing solver. Ambient occlusion was computed at the same time as the solution using the same rendering system.

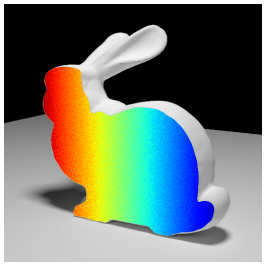
3.3 Results

Generality of WoB

The same WoB framework based on BIEs successfully handles Dirichlet, Neumann, Robin, and mixed boundary problems, for either interior or exterior domains. Figures 1.1 (top-left) and 3.1 show the results of WoB for interior and exterior Laplace problems with known analytical solutions, where we set the boundary conditions to satisfy the known analytical solution. For Robin boundaries, we use a constant value $\overline{\alpha} = 1$ for the mix weight in these examples. For the mixed boundary problems, we uniformly randomly assign one of the Dirichlet, Neumann, or Robin boundary conditions to each boundary triangle or line segment, and use $k = 4$ and $p_k = 2/3$. We show results using the sampling strategies described earlier. For Robin and mixed boundary problems, a pure ray sampling strategy without MIS would miss nonzero contributions, so we instead use RIS with candidates generated uniformly over the boundary to approximately sample the integral kernel. In Fig. 3.1, the number of candidates is 16. All results in Fig. 3.1 are generated with path length $M = 4$ and sample path count per evaluation point $N = 10^7$ for consistency, using a highly parallelized CUDA implementation. We observe that *all* of the results are consistent with the analytical solution as expected.

Convergence Rate and Truncation Error

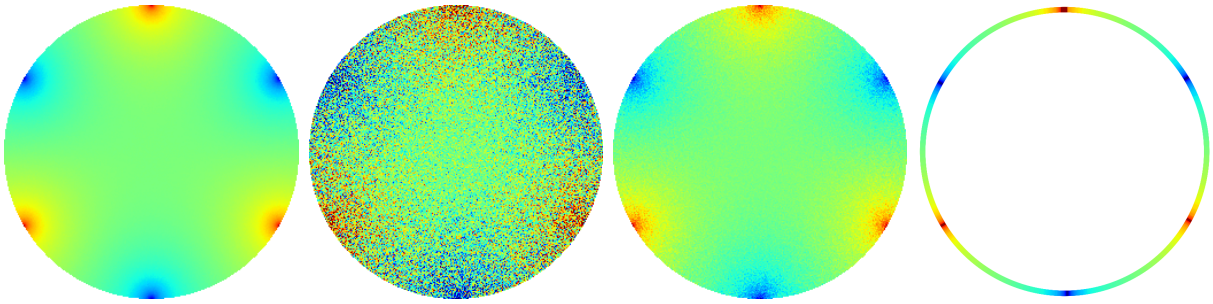
WoB exhibits the expected Monte Carlo convergence rate of $\mathcal{O}(1/\sqrt{N})$. The RMSE curve can eventually become flat when we take enough samples because of the error introduced by path truncation. This error decreases as we increase the path length, although that introduces larger variance to the estimator. For the results in Fig. 3.1, we observe in particular a relatively large truncation error for the two interior Neumann problem estimators and the exterior Dirichlet and mixed boundary problem estimators. We thus show their results with different path lengths in Fig. 3.2. As expected, we observe that having longer paths decreases the truncation error at the cost of having a higher variance, requiring a larger sample count to converge. It might be possible to apply stochastic truncation as in Monte Carlo rendering [Misso et al. 2022] to avoid this error, though a careful investigation is needed to handle subtle differences between WoB and Monte Carlo rendering.



WoB within Monte Carlo Rendering Fig. 3.3 shows the results of WoB implemented within our Monte Carlo ray tracing system. Our implementation of WoB utilizes the existing functionalities of Monte Carlo ray tracing such as ray-object intersection, stochastic sampling, and textures (for boundary values). The code for WoB itself is roughly 100 lines and can support both interior and exterior problems seamlessly. Visualization was done by simultaneously running rendering with ambient occlusion using the Monte Carlo ray tracing system. The inset figure shows a result generated with our prototype interior Dirichlet solver on top of a popular open-source renderer for research purposes, PBRT [Pharr et al. 2018], with minimal modifications. For this problem, we set the boundary values such that we expect to see a linear horizontal gradient of solution values mapped to colors ranging from red to green to blue.

Boundary Sampling and Estimation on the Boundary

WoB for Neumann problems offers two distinct advantages over WoS, as demonstrated in Fig. 3.4. First, we can begin the WoB process by sampling a starting point on the boundary, according to the magnitude of the boundary value, to design an efficient sampling strategy when the boundary value is specified sparsely. With WoS, such a strategy is available only for Dirichlet problems [Qi et al. 2022]. Second, we can exploit the BIE formulation to estimate the solution value exactly on the boundary, with a very minor modification to the underlying BIE. The original WoS and its basic extensions for Neumann problems



(a) reference (b) uniform sampling (c) boundary sampling (d) estimate on boundary

Figure 3.4: Neumann solver with single layer formulation. The boundary value $\partial u/\partial \mathbf{n} = 0$ except for around the six points. By sampling the start of paths according to the distribution of the boundary value (c) in our forward estimator, we get much less noise compared to the one with uniform sampling (b). (a) shows the reference estimate we get with a high sample count. (d) shows the estimate of the solution exactly on the boundary.

that rely on epsilon shell termination criteria are incapable of estimating the solution or the normal derivative exactly on the boundary or exhibit significant bias in such cases.

Gradient Estimator and a Path Reuse Strategy

Similarly to WoS, WoB can estimate the potential and the gradient simultaneously with almost no additional cost. Fig. 3.5 demonstrates the use of WoB for interpolating potential flow velocities, as described by Nielsen and Bridson [2011]. By solving for the gradient of the scalar potential u, which satisfies the Laplace equation and a prescribed Neumann boundary condition, we obtain the velocity field that follows potential flow assumptions (incompressibility and irrotationality) and matches the inflow/outflow conditions at the boundaries. In this example, we use a backward estimator for the single layer formulation with RIS, in combination with a path reuse strategy analogous to the virtual point lights method [Keller 1997] in rendering; we generate sample paths from the boundaries to get samples for the unknown boundary value estimates first, and connect each of them to all the evaluation points to reuse the subpaths starting from the boundaries. This effectively increases the number of sample paths per evaluation point while introducing correlation of the estimates at different evaluation points. Though this correlation may seem undesirable, this approach guarantees that the estimated solution and gradient fields are always smooth, making it a potentially preferable alternative in some settings like fluid simulation

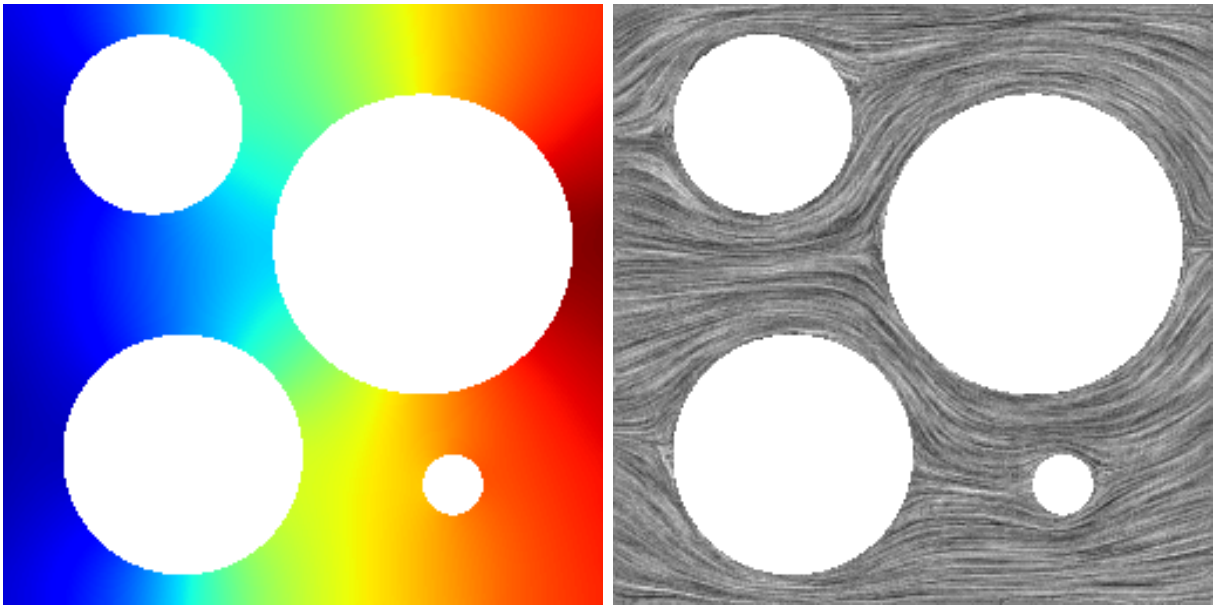


Figure 3.5: Potential flow reconstruction from the velocity boundary condition. We show the estimated potential field on the left and the estimated velocity field on the right. On the outer boundary, a constant inflow boundary condition $\partial u/\partial \mathbf{n} = -1$ is given on the left edge, and a constant outflow boundary condition $\partial u/\partial \mathbf{n} = 1$ is given on the right edge. On the other outer boundary edges and the inner boundaries, $\partial u/\partial \mathbf{n} = 0$ is given.

applications. Miller et al. [2023] introduced a similar boundary value caching technique for WoS in concurrent work, and their analysis largely applies to the case of WoB as well.

MIS Estimator

Due to the similarity between WoB and Monte Carlo rendering, it is trivial to combine WoB estimators via MIS. Fig. 3.6 compares the backward estimator, the backward estimator with importance sampling of boundary values, and the MIS combination of the two estimators with the balance heuristic for a Dirichlet problem. The two estimators correspond to unidirectional path tracing and path tracing with next-event estimation in Monte Carlo ray tracing. When the non-zero boundary values are not localized (top row), the backward estimator performs well, although boundary sampling (which corresponds to next-event estimation) suffers from additional noise due to its explicit connection to a boundary point. Conversely, when the non-zero boundary values are localized (bottom row), boundary sampling becomes significantly more efficient than the backward estimator. This behavior is analogous to unidirectional path tracing and next-event estimation for cases where light sources are large or small. The MIS combination of the two estimators (right column) is robust across different settings. The related work for WoS [Qi et al. 2022] left this MIS combination to future work, as the integration domain changes at each step in WoS (i.e., WoS solves a Volterra equation; see Section 3.4.1). WoB allows us to incorporate MIS since it has the same mathematical and algorithmic structure as Monte Carlo ray tracing. Notably, our bidirectional estimators for WoB also do not introduce extra bias, unlike their WoS counterparts [Qi et al. 2022].

Markov Chain Monte Carlo

We implemented Primary Sample Space Metropolis Light Transport [Kelemen et al. 2002] (PSSMLT) on top of WoB interior/exterior Dirichlet estimators. In this implementation, each sample in WoB is generated according to a Markov chain over the sampling domain, including the image space. PSSMLT formulates this sampling domain as a unit hypercube of random numbers used to generate each sample and the location in the image space. In our case, the first two dimensions are used to pick a pixel (i.e., evaluation point) and the rest of the dimensions are used for generating a sample in the corresponding WoB estimator (i.e., interior or exterior, depending on the pixel), just like PSSMLT in rendering. We set the target distribution to be the absolute value of the sample's contribution, so samples that have higher absolute contributions are likely to be generated more often. Since this target distribution includes all the terms in each sample (e.g., the boundary condition and

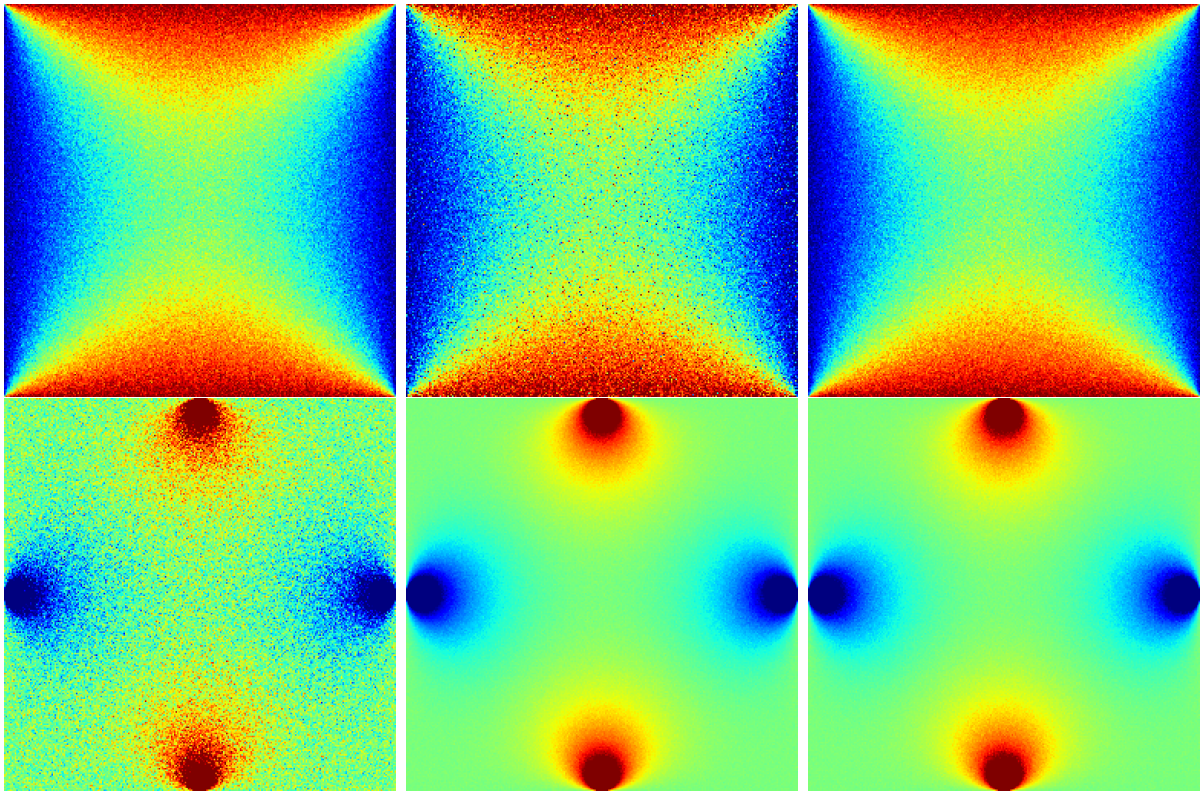


Figure 3.6: Equal-time comparison of bidirectional WoB estimators. Left to right: purely backward estimator, next-event estimation, and the MIS combination of the two. Top row: non-zero boundary values on each side. Bottom row: non-zero boundary values around the center of each side. Either the backward estimator or next-event estimation is more efficient than the other in each setting. WoB allows us to trivially combine the two estimators via MIS, and the combined estimator is robust across different settings (right).

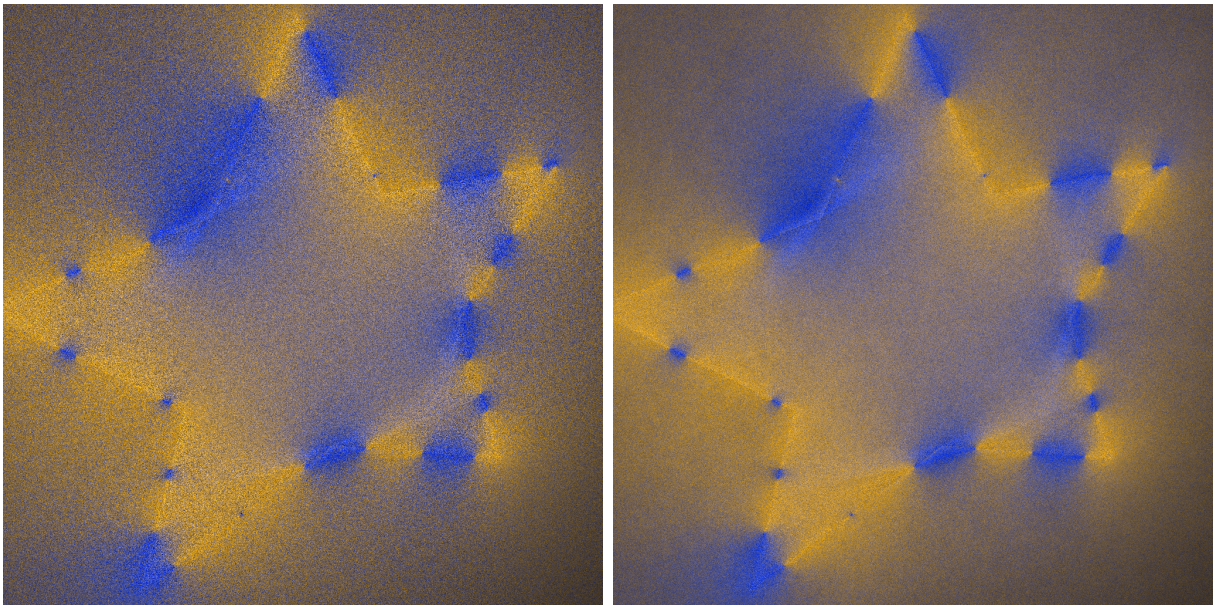


Figure 3.7: Equal-time comparison of Monte Carlo and Markov chain Monte Carlo (MCMC) estimators for solving interior and exterior Dirichlet problems on a plane passing through a concave 3D star shape. Left image: Monte Carlo sampling. Right image: Markov chain Monte Carlo sampling formulated as PSSMLT [Kelemen et al. 2002]. MCMC can result in lower variance than the standard Monte Carlo method, as in rendering.

the probability of selection in all-hits), MCMC is expected to perform better than the standard Monte Carlo method with a limited form of importance sampling. Fig. 3.7 shows our preliminary examples that compare the standard Monte Carlo method and MCMC in equal time. These examples solve interior and exterior Dirichlet problems at the same time. We observe that PSSMLT performs similarly in rendering and WoB in the sense that the image is less noisy than the standard Monte Carlo method at the cost of correlation artifacts. This application of MCMC is straightforward due to the similarity between rendering and WoB.

Numerical Comparisons between WoB and WoS Fig. 3.8 compares the efficiency of WoB and WoS interior Dirichlet estimators in examples of convex and non-convex domains. Both WoB and WoS are implemented with similarly optimized CUDA code in their basic forms with no advanced sampling techniques. Our experiments suggest that WoS performs more efficiently than WoB for complex non-convex domains, but comparably for simple

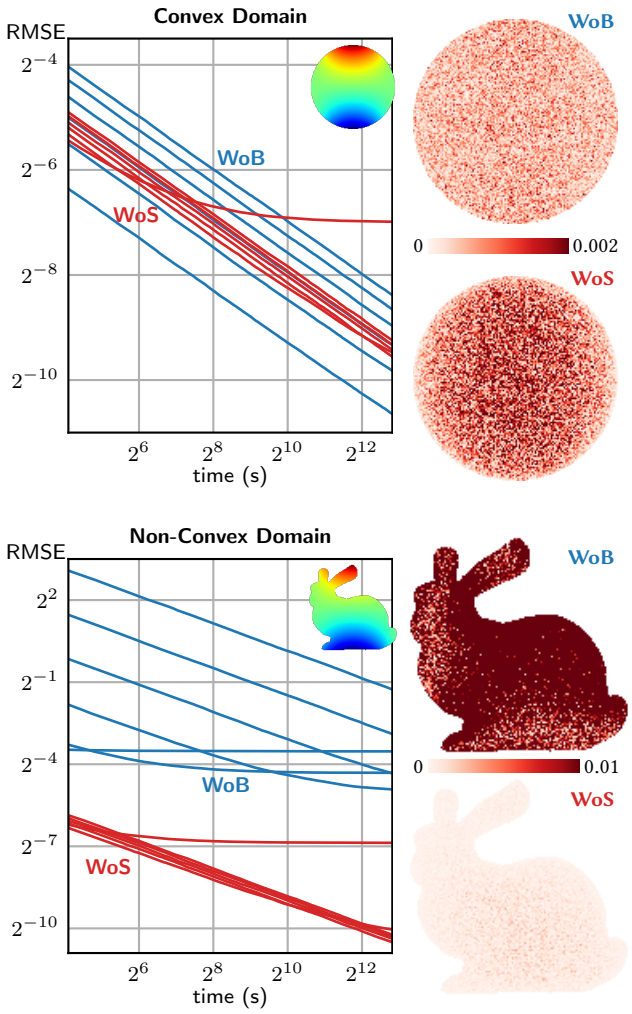


Figure 3.8: Comparison of WoB and WoS with example interior Dirichlet problems in a convex domain (top) and a non-convex domain (bottom). The plots show how the root mean squared errors (vertical axis) decay with increasing time (horizontal axis). Each line corresponds to a specific parameter choice for WoB (blue) and WoS (red). We use path length $M = 2$ to 7 for WoB, and epsilon shell size 10^{-2} to 10^{-7} for WoS. For each scene and for each method, we show the error of the solution with the least error after 2 hours on the right. We observe that WoS performs more efficiently than WoB for complex non-convex domains.

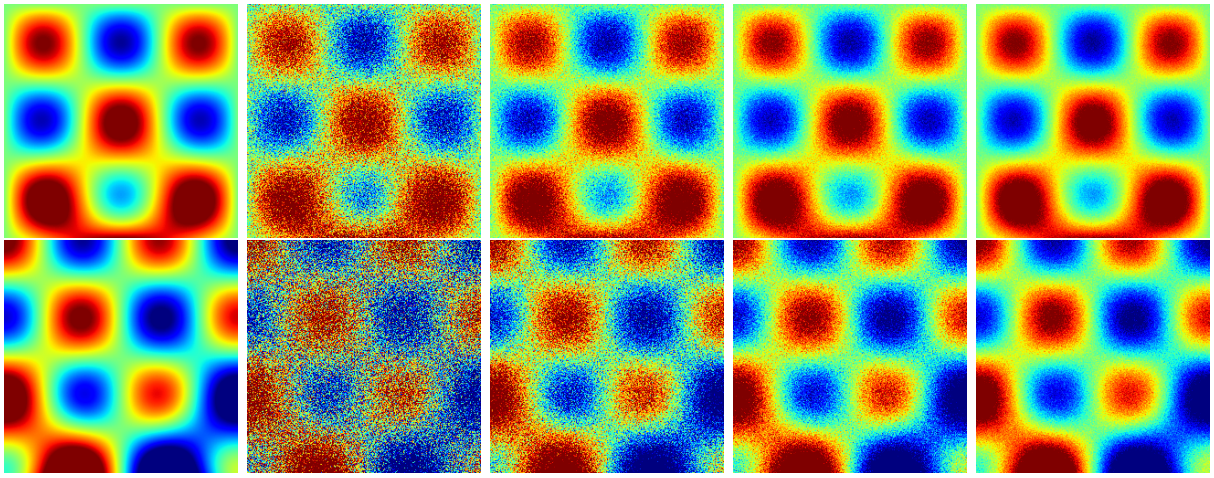


Figure 3.9: Estimates for the Poisson equation for Dirichlet (top) and Neumann (bottom) problems. WoB can handle the non-zero source term. For each sample path, we used 16 volume samples to estimate all domain integrals.

convex domains. The reason is that, while WoS typically takes more computation time per sample (i.e., per individual random walk that terminates near the boundary), WoB requires more sample paths due to its higher variance, resulting in lower overall efficiency. We expect that this difference may become smaller with additional variance reduction techniques. Moreover, this observation only provides general guidance on the choice between WoS and WoB. We do not claim that one of them is fundamentally more efficient than the other in any given problem (be it convex or non-convex) based on these numerical comparisons; the theoretical comparisons offered throughout this paper are more generally relevant (e.g., accuracy near the boundary and generality regarding supported problems).

3.4 Discussion

3.4.1 Classes of Integral Equations

Rendering equation, WoS, and WoB

Both the rendering equation and the formulation of WoS are understood to be Fredholm equations of the second kind [Pharr et al. 2018; Qi et al. 2022]. This type of integral

equation takes the form $f(\mathbf{x}) = g(\mathbf{x}) + \int_D K(\mathbf{x}, \mathbf{y}) f(\mathbf{y})\, d\mathbf{y}$ where $K(\mathbf{x}, \mathbf{y})$ is a given integral kernel, $g(\mathbf{x})$ is a known function, $f(\mathbf{y})$ is an unknown function we want to solve for, and D is a fixed integration domain. When the integration domain changes depending on $\mathbf{x}$, it is called a Volterra equation (of the second kind). Because we can rewrite any Volterra equation into a form that has a fixed integration domain by multiplying the integrand by an additional indicator function, any Volterra equation is a Fredholm equation. Still, we will use these terms in a narrow sense without considering such conversions here.

In rendering, for surface light transport, $\mathbf{x}$ is a tuple of a location on the surface and a direction from there (i.e., to measure radiance coming from that particular location toward the particular direction). The function $g(\mathbf{x})$ is the emission term, and $K(\mathbf{x}, \mathbf{y})$ is defined as $K(\mathbf{x}, \mathbf{y}) = f_r(\mathbf{x}, \mathbf{y}) V(\mathbf{x}, \mathbf{y}) G_{eo}(\mathbf{x}, \mathbf{y})$ where f_r is the BSDF, V is the visibility term, and G_{eo} is the geometry term [Pharr et al. 2018]. The integration domain is fixed as the surfaces of the scene, so it fits the definition of Fredholm equations.

In WoS, $g(\mathbf{x})$ is defined as $g(\mathbf{x}) = \int_{B_R(\mathbf{x})} \overline{b}(\mathbf{z}) G_R(\mathbf{x}, \mathbf{z}) d\mathbf{z}$ where $\overline{b}(\mathbf{x})$ is the source function, G_R is Green's function for the largest ball $B_R(\mathbf{x})$ contained within the domain centered at $\mathbf{x}$. Note that both functions are given so $g(\mathbf{x})$ is also still given. The kernel $K(\mathbf{x}, \mathbf{y})$ for WoS is defined as $K(\mathbf{x}, \mathbf{y}) = \frac{\partial G_R(\mathbf{x}, \mathbf{y})}{\partial \mathbf{n}_\mathbf{y}}$ and the integration domain is $D = \partial B_R(\mathbf{x})$, the surface of the ball $B_R(\mathbf{x})$. While Qi et al. [2022] claimed that the formulation of WoS is a Fredholm equation of the second kind, it would be more natural to call it a Volterra equation of the second kind since the integration domain $D = \partial B_R(\mathbf{x})$ changes according to $\mathbf{x}$. A connection between WoS and the rendering equation was imperfectly made in this sense. On the other hand, in WoB, the integration domain is fixed as the boundary, so it is precisely a Fredholm equation.

The rendering equation, commonly referred to as a Fredholm equation [Pharr et al. 2018], appears to contradict the definition of a fixed integration domain when it is solved by Monte Carlo ray tracing. Ray tracing from a point $\mathbf{x}$ results in a varying set of visible points $\mathbf{y}$ depending on $\mathbf{x}$, and thus it appears to be a Volterra equation. However, the integration domain is fixed as the surfaces of the scene, so it still fits the definition of a Fredholm equation. While this mismatch is paradoxical, we have identified that classifying the rendering equation as a Fredholm equation can be misleading.

In addition to the aforementioned area form, the rendering equation can take a solid angle form, in which the integration domain D becomes the hemispherical angular domain around $\mathbf{x}$, and the kernel $K(\mathbf{x}, \mathbf{y})$ is defined as $f_r(\mathbf{x} \to \mathbf{y}) \cos\theta$, where $\mathbf{y}$ is the first visible point from $\mathbf{x}$ along the direction towards $\mathbf{y}$ [Pharr et al. 2018]. This solid-angle form is the form used in Monte Carlo ray tracing, and in this context, the rendering equation is a Volterra equation of the second kind, due to the changing angular integration domain

based on $\mathbf{x}$. However, the area form of the rendering equation, which integrates over all surface points, is a Fredholm integral equation of the second kind. The assertions made by Qi et al. [2022] still hold if one accepts that the solid-angle form of the rendering equation is a Volterra equation, just like the formulation of WoS.

As discussed earlier, this distinction is subtle since a Volterra equation can be converted into a Fredholm equation by properly expanding the kernel $K(\mathbf{x}, \mathbf{y})$ with zeros in a common fixed integration domain. This conversion is precisely what actually occurs in the area form of the rendering equation, where the visibility term $V(\mathbf{x}, \mathbf{y})$ returns zero for points $\mathbf{y}$ that are not visible from $\mathbf{x}$. The area form of the rendering equation is still a Fredholm equation, but it can be transformed into a Volterra equation by redefining the integration domain to include only visible points from $\mathbf{x}$. The formulation of WoB cannot be reduced to a Volterra equation, as its kernel is nonzero everywhere.

Singularity and Reciprocity of the Kernel

BIEs involve singular kernels, where the kernel becomes unbounded as the distance between points $r = |\mathbf{x} - \mathbf{y}|$ approaches zero. The order of singularity in the kernel, K, can be classified as weakly singular, strongly singular, and hypersingular, for $K = \mathcal{O}(1/r)$, $K = \mathcal{O}(1/r^2)$, and $K = \mathcal{O}(1/r^3)$ respectively. The orders of singularity here pertain to three-dimensional scenarios, although the underlying concepts remain unchanged in two-dimensional scenarios. As the order of singularity increases, robust estimation becomes challenging. Weak and strong singularities in the kernel are prevalent in the rendering equation, and a plethora of techniques to circumvent numerical difficulties have been studied [Pharr et al. 2018]. For instance, tracing a ray, as opposed to directly sampling a point on surfaces, can avoid singularities arising from the geometry term. Similar methods are applied to WoB. However, hypersingular integrals necessitate special attention during computation and are not typically encountered in rendering. It is advisable to avoid high-order singularities whenever possible.

The kernel K is called symmetric if $K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x})$ and asymmetric otherwise. In rendering, the terms reciprocal and non-reciprocal are used instead of symmetric and asymmetric, respectively; the kernel of the rendering equation is usually symmetric due to the physics of light, but can be asymmetric in certain cases [Veach 1998]. Both WoB and WoS deal with asymmetric kernels but they do not pose a problem for the solvers as long as they are treated properly, similarly to handling of asymmetric kernels in rendering.

First-Kind Equations and Their Monte Carlo Estimation

Both Fredholm and Volterra integral equations have first- and second-kind forms. A Fredholm equation of the first kind takes the form $g(\mathbf{x}) = \int_D K(\mathbf{x}, \mathbf{y}) f(\mathbf{y})\, d\mathbf{y}$, where both g and K are known and f is the unknown function to be solved for. Note we used the term first kind equation abusively in the main text: strictly, we should call it a first kind equation if the equation holds for the entire integral domain, but in our mixed boundary problem estimator, we have different integral equations defined conditionally on the type of boundary at each point. There is also a third kind of Fredholm equation, but it is not relevant to WoB in our paper.

While second-kind equations can be estimated using Monte Carlo integration via Neumann series expansion, this technique is not applicable to first-kind equations. For instance, one might consider estimating the integral using Monte Carlo integration as: $g(\mathbf{x}) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{K(\mathbf{x}, \mathbf{y}_i) f(\mathbf{y}_i)}{p(\mathbf{y}_i)}$. This equation cannot be used, as f is unknown and cannot be solved for using Monte Carlo integration alone. We therefore focused on having second-kind equations that are compatible with Monte Carlo integration, which requires us to choose a specific BIE to achieve this goal.

3.4.2 WoB vs. WoS(t)

WoB supports a broader range of boundary conditions than the baseline WoS [Muller 1956; Sawhney and Crane 2020], and its similarity to Monte Carlo rendering enables the application of advanced variance reduction and sampling techniques to PDEs. While the concurrently proposed WoSt method [Sawhney and Miller et al. 2023, 2024b] extends the applicability of WoS-based methods, it introduces two key challenges already discussed in Section 2.3.4: handling exterior domains necessitates nontrivial modifications to the algorithm, and the required geometric queries can be difficult to implement for general boundary representations.

That said, we acknowledge that the WoS(t) approach tends to demonstrate higher efficiency for interior problems with nonconvex domains compared to the current implementation of WoB; examples for Dirichlet problems can be seen in Fig. 3.8. The bottom-up design of WoS(t) tightly couples the choice of integral equation and the sampling technique to maximize efficiency. In particular, WoS(t) avoids the need to sample all ray-boundary intersection points, a key bottleneck in our WoB method.

However, this design philosophy is fundamentally different from WoB, which serves as a general framework. Unlike WoS(t), which leaves relatively limited room for structural

changes, WoB provides a flexible foundation for further development. The implementations presented in this thesis are only specific instances of WoB, and many opportunities remain for improvement, including the selection of integral equations, path truncation strategies, and sampling techniques. We will discuss several concrete directions in Chapter 7. In particular, for the convex domain problem in Fig. 3.8, with short path lengths, we observed a superior performance of WoB, which highlights its potential and motivates future work on improving its efficiency more broadly.

Chapter 4

Monte Carlo PDE Solvers for Parabolic or Vector-Valued Problems

In the previous chapter, we discussed the formulation and the advantages of the Walk on Boundary (WoB) Monte Carlo PDE solver for the Laplace and Poisson equations. These equations are elliptic and not time-dependent. With these equations, we examined the WoB solver when the unknown variables of our primary interest, along with all quantities in their formulation, such as kernel functions and unknown boundary density functions, are scalar-valued. Due to this time-independence and scalar nature, the formulation of WoB for the Laplace and Poisson equations represented its simplest form. This chapter explains how WoB can be readily extended to time-dependent parabolic equations and vector-valued equations, using the diffusion and Stokes equations as examples. As with the Laplace equation, WoB performs a sampling-based evaluation of boundary integral equations (BIEs), enabling highly parallel pointwise evaluation of the solution. In other words, by construction, these methods inherit the same key properties as WoB when applied to the Laplace equation.

4.1 Monte Carlo PDE Solver for Diffusion Equation

To understand the broader applicability of WoB, we first discuss its generalization to *time-dependent* PDEs. These typically fall into two main categories: *parabolic* and *hyperbolic* PDEs. Roughly speaking, parabolic PDEs describe phenomena related to the diffusion of quantities, for instance, heat propagation. They model the propagation of these quantities

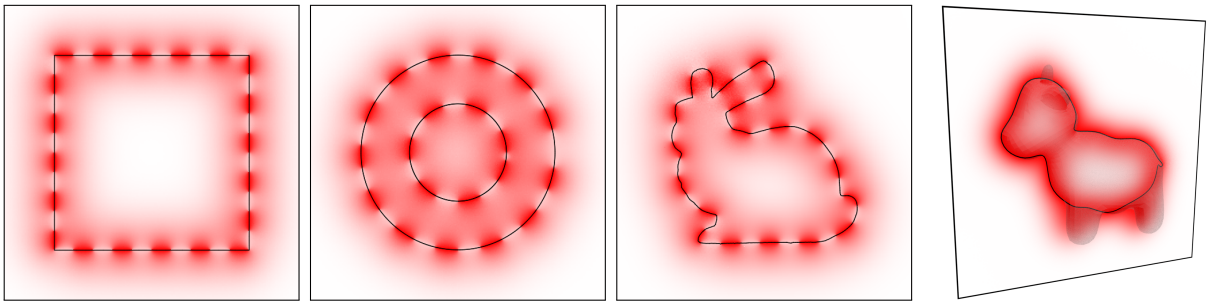


Figure 4.1: Monte Carlo method for the diffusion equation. We solve the diffusion equation using WoB. The results shown evaluate the solution at a specific time on grid points across various domains, with domain boundaries shown in black. The left three are 2D results, while the rightmost shows a 3D result evaluated on a 2D slice. This method enables *pointwise* estimation of the solution in *both space and time*.

as changes in the underlying field that propagate at infinite speed, with effects decaying exponentially in space, leading to a smoothing effect over time. Conversely, hyperbolic PDEs describe wave propagation at finite speed, maintaining sharp features. Both types require the specification of an *initial condition* in addition to boundary conditions, resulting in initial-boundary value problems, a contrast to elliptic equations, which only demand boundary conditions. To this end, we focus on WoB for the *diffusion equation* with Dirichlet boundary conditions (Fig. 4.1), as originally presented by Sabelfeld [1991, Chapter 5]. While we do not discuss the specifics here, it is critically important to note that this method generalizes effectively to other types of boundary conditions (e.g., Neumann), as demonstrated by Sabelfeld [1991]. As our original contribution, we will later present a numerical study of the method in Section 4.3.1, which has not been provided in depth in the literature. While we focus on the diffusion equation, the core idea would extend to other parabolic PDEs. In contrast to parabolic PDEs, the application of WoB or any Monte Carlo methods to hyperbolic PDEs in general scenarios remains largely unexplored, and we will defer the discussion to Section 7.2.1.

4.1.1 Diffusion Equation and Its Fundamental Solution

The equation we will consider is the homogeneous, constant-coefficient diffusion equation,

$$\frac{\partial u(\mathbf{x},t)}{\partial t} = \alpha \nabla^2 u(\mathbf{x},t) \quad \text{for } \mathbf{x} \in \Omega, t > 0, \tag{4.1}$$

where $\alpha > 0$ is a constant diffusion coefficient. This equation is sometimes referred to as the heat equation as well. The boundary condition we consider is the Dirichlet boundary condition,

$$u(\mathbf{x}, t) = \overline{u}_D(\mathbf{x}, t) \quad \text{for } \mathbf{x} \in \partial\Omega, t > 0, \tag{4.2}$$

where $\overline{u}_D$ is a prescribed, time-dependent function defined over the boundary $\partial\Omega$. In addition to the boundary condition, we must specify the initial condition,

$$u(\mathbf{x}, 0) = \overline{u}_0(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Omega, \tag{4.3}$$

where $\overline{u}_0$ is a prescribed function defined over the entire domain Ω. We do not need to specify $\frac{\partial u}{\partial t}$ at $t = 0$ because it is implicitly given by Eq. 4.1 once we specify Eq. 4.3.

By introducing the change of variables $s = \alpha t$ and $v(\mathbf{x}, s) = u(\mathbf{x}, t/\alpha)$, one can rewrite the diffusion equation (Eq. 4.1) with an arbitrary constant diffusion coefficient and the corresponding boundary and initial conditions (Equations 4.2 and 4.3) into an equivalent one with a unit diffusion coefficient:

$$\begin{aligned} \frac{\partial v(\mathbf{x}, s)}{\partial s} &= \nabla^2 v(\mathbf{x}, s) && \text{for } \mathbf{x} \in \Omega, s > 0, \\ v(\mathbf{x}, s) &= \overline{u}_D(\mathbf{x}, s/\alpha) && \text{for } \mathbf{x} \in \partial\Omega, s > 0, \text{ and} \\ v(\mathbf{x}, 0) &= \overline{u}_0(\mathbf{x}) && \text{for } \mathbf{x} \in \Omega. \end{aligned} \tag{4.4}$$

Therefore, without loss of generality, we will assume a unit diffusion coefficient throughout the remainder of this work.

Now, we introduce the fundamental solution to the diffusion equation, also known as the heat kernel. The fundamental solution $Z(\mathbf{x}, t; \mathbf{y}, \tau)$ to the heat equation is defined as the response $u(\mathbf{x}, \tau)$ to a Dirac delta source located at point $\mathbf{y}$ and time τ in an infinite domain:

$$\begin{aligned} \frac{\partial u(\mathbf{x}, t)}{\partial t} &= \nabla^2 u(\mathbf{x}, t) && \text{for } \mathbf{x} \in \mathbb{R}^d, t > \tau, \text{and} \\ u(\mathbf{x}, \tau) &= \delta(\mathbf{x} - \mathbf{y}) && \text{for } \mathbf{x} \in \mathbb{R}^d \end{aligned} \tag{4.5}$$

and is given as

$$Z(\mathbf{x}, t; \mathbf{y}, \tau) = \Theta(t')(4\pi t')^{-d/2} e^{-r^2/4t'}, \tag{4.6}$$

where $r = \|\mathbf{y} - \mathbf{x}\|_2$, $t' = t - \tau$, and $\Theta(\cdot)$ is the Heaviside step function.

Analogous to the Poisson equation with a nonzero source function, we can solve the diffusion equation in the infinite space $\mathbb{R}^d$ with an arbitrary initial condition by the convolution of the initial condition with the fundamental solution. The solution to Eq. 4.1 with

initial condition Eq. 4.3 when $\Omega = \mathbb{R}^d$ is

$$u(\mathbf{x}, t) = \int_{\mathbb{R}^d} Z(\mathbf{x}, t; \mathbf{y}, 0)\overline{u}_0(\mathbf{y})\,\mathrm{d}\mathbf{y}. \tag{4.7}$$

Thus, when the domain is the infinite domain, we can see that we can apply Monte Carlo integration or quadrature rules to estimate this integral to get a numerical solution to the problem. WoB will generalize this to problems with boundaries.

4.1.2 BIEs for the Diffusion Equation

We aim to develop a Monte Carlo solver for the diffusion equation with Dirichlet boundary conditions within an interior or exterior domain Ω. Similarly to the case of the Poisson equation, we can write the solution to the diffusion equation in the form of the double layer potential and an additional initial condition term for $\mathbf{x} \in \Omega$ and $t > 0$:

$$\begin{aligned} u(\mathbf{x}, t) = &- \int_0^t \int_{\partial\Omega} \frac{\partial Z}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}, t;\, \mathbf{y}, \tau)\phi(\mathbf{y}, \tau)\,\mathrm{d}\mathbf{y}\,\mathrm{d}\tau \\ &+ \int_\Omega Z(\mathbf{x}, t; \mathbf{y}, 0)\overline{u}_0(\mathbf{x}, 0)\,\mathrm{d}\mathbf{y}, \end{aligned} \tag{4.8}$$

where $\mathbf{n}_\mathbf{y}$ is the normal pointing outward from the domain Ω, and $\phi(\mathbf{y}, \tau)$ is an unknown density function defined for $\mathbf{y} \in \partial\Omega$ and $\tau > 0$. We can also derive a recursive BIE for $\phi(\cdot, \cdot)$ by taking the limit $\mathbf{x} \to \partial\Omega$ in Eq. 4.8 and reordering terms:

$$\begin{aligned} \phi(\mathbf{x}, t) = &\int_0^t \int_{\partial\Omega} 2\frac{\partial Z}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x}, t;\, \mathbf{y}, \tau)\phi(\mathbf{y}, \tau)\,\mathrm{d}\mathbf{y}\,\mathrm{d}\tau \\ &+ 2\overline{u}_D(\mathbf{x}, t) - 2\int_\Omega Z(\mathbf{x}, t; \mathbf{y}, 0)\overline{u}_0(\mathbf{x}, 0)\,\mathrm{d}\mathbf{y}. \end{aligned} \tag{4.9}$$

Note that the integral equation here has an unknown ϕ that depends on both the time variable t and the space variable $\mathbf{x}$. Moreover, on the right-hand side, the integral domain for the time variable τ depends on the current time t; this integral equation is therefore typically classified as the Volterra equation of the second kind. While it is possible to extend the integrand by zero outside the original domain and treat the problem similarly to other Fredholm equations of the second kind, designing a sampling scheme that restricts samples to the original integration domain is expected to produce an estimator with lower variance.

Note also that the definition of normal directions here differs from the normal directions we used in the previous chapters in their signs, so we can describe both the interior and exterior problems with a single pair of equations. This convention is used in this chapter and in the next chapter.

4.1.3 Walk on Boundary Method for the Diffusion Equation

Based on the above integral equations, Sabelfeld [1991, Chapter 5] designed WoB for the diffusion equation, which we describe here, similarly to the design of the method for the Laplace equation. This method still exhibits very similar properties to those we have observed for the Laplace equation; we perform a highly parallelizable, pointwise estimation of the solution using ray intersection sampling.

Backward Estimator

We design a backward Monte Carlo estimator for the diffusion equation based on Eq. 4.8 and Eq. 4.9. First, we define an N_I-sample estimator for the initial condition term (the second term in Eq. 4.8) with PDF p_I as

$$\langle I(\mathbf{x},t)\rangle = \frac{1}{N_I}\sum_{i=1}^{N_I}\frac{Z(\mathbf{x},t;\mathbf{y}^i,0)}{p_I(\mathbf{y}^i|\mathbf{x},t)}\overline{u}_0(\mathbf{y}^i,0). \tag{4.10}$$

Using this estimator, we can define an estimator for Eq. 4.8 with PDF p_D as

$$\langle u(\mathbf{x},t)\rangle = \left\{\frac{1}{N_D}\sum_{j=1}^{N_D}-\frac{\frac{\partial Z}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x},t;\,\mathbf{y}^j,\tau^j)}{p_D(\mathbf{y}^j,\tau^j|\mathbf{x},t)}\langle\phi(\mathbf{y}^j,\tau^j)\rangle\right\} + \langle I(\mathbf{x},t)\rangle \tag{4.11}$$

and an estimator for Eq. 4.9 with PDF p_E as

$$\langle\phi(\mathbf{x},t)\rangle = 2\frac{\frac{\partial Z}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x},t;\,\mathbf{y},\tau)}{p_E(\mathbf{y},\tau|\mathbf{x},t)}\langle\phi(\mathbf{y},\tau)\rangle + 2\overline{u}_D(\mathbf{x},t) - 2\langle I(\mathbf{x},t)\rangle. \tag{4.12}$$

Both of these require a recursive estimation of $\langle\phi(\mathbf{y},\tau)\rangle$ included on the right-hand side of the equations. To estimate the solution at $(\bar{\mathbf{x}},\bar{t})$, we use Eq. 4.11. Then, we apply Eq. 4.12 recursively by generating a sequence of points in $\partial\Omega\times(0,\bar{t})$ moving backward in time toward the initial time. Note that the PDFs p_D and p_E only need to sample points with time $\tau < t$ to get an unbiased estimate, since for $\tau \geq t$, the integral kernel $\frac{\partial Z}{\partial \mathbf{n}_\mathbf{y}}$ is zero

by definition and the integrand evaluates to zero. Intuitively, this reflects the causality of the diffusion equation, where the solution at time t depends only on prior times, not future ones. Utilizing this property, we can choose to sample points in the space-time domain such that the times in the sampled sequence strictly decrease. In the sampling strategy discussed later, we sample the time τ with a probability that decreases as a function of the difference between the current time t and the sample time τ, allowing us to sample any negative times as well. The recursion terminates once the sampled time τ becomes negative, as the original integral domain excludes negative times. While WoB for the Poisson equation introduces bias due to path truncation, the corresponding method for the diffusion equation yields an *unbiased* solution estimate because there is no arbitrary path truncation. See the analysis by Sabelfeld [1991] for a more formal argument.

Sampling

For the initial condition sampling $p_I(\mathbf{y}^i|\mathbf{x},t)$, we sample the points by $\mathbf{y}_i \leftarrow \mathbf{x} + 2\sqrt{t\gamma_{d/2}}\omega$, where $\gamma_{d/2}$ is a sample drawn from the Gamma distribution with shape parameter $d/2$ and scale parameter 1, and ω is a uniformly random direction sampled on a unit sphere. To draw samples from the Gamma distribution, we use $\gamma_1 \leftarrow -\ln(\alpha)$ for 2D, where α is a uniformly random sample in the range $(0,1)$, and $\gamma_{3/2} \leftarrow \gamma_1 + \xi^2/2$ for 3D, where ξ is a standard normal sample. With this sampling strategy, we get

$$\frac{Z(\mathbf{x},t;\mathbf{y}^i,0)}{p_I(\mathbf{y}^i|\mathbf{x},t)} = 1, \tag{4.13}$$

importance sampling according to the kernel function. With this sampling strategy, we can end up sampling points that are not contained within the domain of interest Ω. To deal with this case, we extend the integrand to be zero outside Ω and let the samples falling into this part have zero contribution to have an unbiased estimate of the initial condition term.

For $p_D(\mathbf{y},\tau|\mathbf{x},s)$ and $p_E(\mathbf{y},\tau|\mathbf{x},s)$, we first sample the spatial point $\mathbf{y}$ using the uniform line intersection sampling as in Section 3.2.4, and then sample time τ, given t, by $\tau \leftarrow t - \frac{\|\mathbf{y}-\mathbf{x}\|^2}{4\gamma_{d/2}}$. When the sampled time τ is of a nonnegative time,

$$\begin{aligned} \frac{\frac{\partial Z}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x},t;\,\mathbf{y},\tau)}{p_D(\mathbf{y},\tau|\mathbf{x},t)} &= -\kappa(\mathbf{x},\mathbf{y})\cdot \operatorname{sgn}(\mathbf{n}(\mathbf{y})\cdot(\mathbf{y}-\mathbf{x})) \quad \text{for } \mathbf{x}\in\Omega \quad \text{, and} \\ 2\frac{\frac{\partial Z}{\partial \mathbf{n}_\mathbf{y}}(\mathbf{x},t;\,\mathbf{y},\tau)}{p_E(\mathbf{y},\tau|\mathbf{x},t)} &= -\kappa(\mathbf{x},\mathbf{y})\cdot \operatorname{sgn}(\mathbf{n}(\mathbf{y})\cdot(\mathbf{y}-\mathbf{x})) \quad \text{for } \mathbf{x}\in\partial\Omega, \end{aligned} \tag{4.14}$$

where sgn is the sign function and $\kappa(\mathbf{y}, \mathbf{x})$ is the number of intersection points that the line that goes through the two points has, excluding point $\mathbf{x}$. Similar to Section 3.2.4, this sampling strategy samples points perfectly proportionally to the kernel functions. We let these two estimates be 0 when $\tau < 0$, as the normal derivative of the fundamental solution contained in the integrand equals zero for $\tau < 0$. In the above, the left-hand side scaling factors of the two expressions differ by 2, but the right-hand sides have the same expression. This is because the two PDFs p_D and p_E differ even though we use the same strategy: point $\mathbf{x}$ for p_D lies within the domain, whereas point $\mathbf{x}$ for p_E lies on a boundary.

When we sample the diffusion equation with the above sampling scheme, we notice that this algorithm is a generalization of WoB for the Laplace equation, in the sense that we sample points in the spatial domain in the same way and assign them signed weights in the same way as in Section 3.2.4. However, for the diffusion equation, we additionally keep track of the time associated with each sample and truncate the paths when the associated time goes negative. As the sequence of samples generated with this strategy has a strictly decreasing time sequence, all paths would terminate naturally without relying on somewhat arbitrary path truncation. Thus, this method yields unbiased estimates of the solution with finite variances. For a more rigorous discussion, see the analysis by Sabelfeld [1991, Section 5.1.2].

4.2 Monte Carlo PDE Solver for Stokes Equations

The other equations we consider in this section are the *Stokes equations*, which describe the flow of highly viscous, incompressible fluids. These are elliptic and thus time-independent, but unlike the *scalar-valued* Laplace or Poisson equations, their solutions are *vector-valued*. The application of WoB to Stokes equations is mostly straightforward; we will derive BIEs and approximate their solutions using recursive Monte Carlo integration. To study the behavior of the numerical solutions, we consider a benchmark for the Stokes equations: the Taylor–Couette flow (Fig. 4.2). The derivation here presents the application of WoB to the Stokes equations.

4.2.1 Stokes Equations and Their Fundamental Solutions

In a bounded or unbounded domain Ω, the Stokes equations are given by

$$\nabla p - \mu \nabla^2 \mathbf{u} = \mathbf{f}, \quad \text{and} \quad \nabla \cdot \mathbf{u} = 0, \tag{4.15}$$

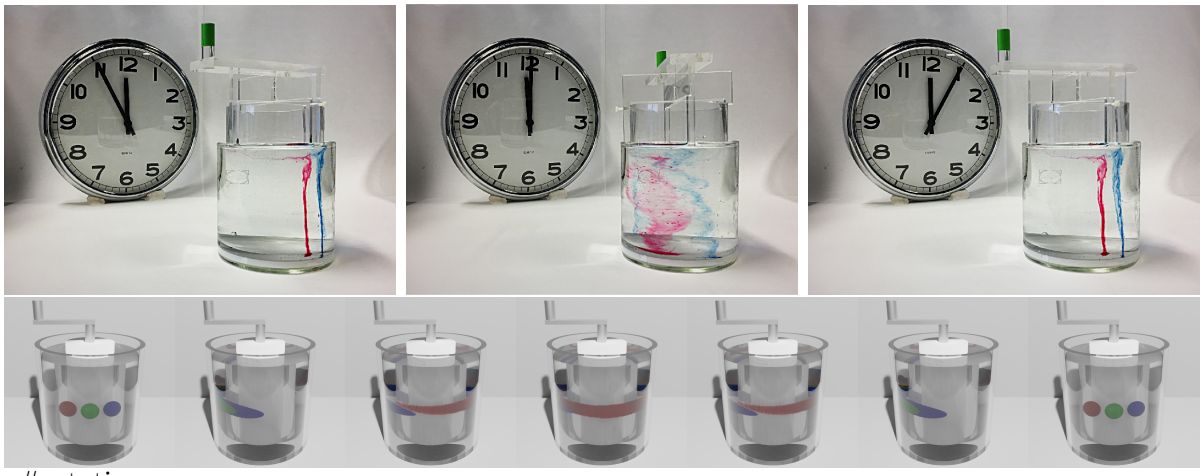


Figure 4.2: Frames (left to rignt) from real-world footage of 3D Taylor-Couette flow (top) alongside the corresponding simulation (bottom) performed by solving the Stokes equations with mixed boundary conditions using WoB. A body of highly viscous liquid is contained within a domain bounded by inner and outer cylinders. As the inner cylinder is rotated first in one direction and then slowly rotated back (from left to right), the liquid roughly returns to its original configuration, as seen by the ink's position in the footage. In the WoB simulation, the inner cylinder is rotated three full turns clockwise and then reversed. The numbers at the bottom indicate the cumulative number of rotations of the inner cylinder from the initial position. As intended, the simulation successfully returns the ink drops, modeled as marker particles, to their original positions after completing the full sequence. (real-world images by Vid1123/Wikimedia CC BY-SA 4.0)

where p is the pressure, $\mathbf{u}$ is the velocity, μ is the dynamic viscosity coefficient, and $\mathbf{f}$ is a body force. The first equation represents momentum conservation, while the second enforces incompressibility.

The full set of equations governing general fluid motion, the Navier–Stokes equations, is nonlinear. We will discuss this more complex case in Chapter 5. The Stokes equations in Eq. 4.15 arise as a simplification of the Navier–Stokes equations under the assumption that inertial effects are negligible and the flow is incompressible. This approximation is valid in regimes with high viscosity and low flow speed, that is, at low Reynolds numbers.

The system in Eq. 4.15 constitutes a set of linear elliptic partial differential equations, similar in nature to Laplace's equation. As such, it has been extensively studied, including through boundary integral formulations and various discretization-based solvers [Abousleiman and Cheng 1994; Alves et al. 2021; Power and Wrobel 1995]. A related system is the unsteady Stokes equations, which augment Eq. 4.15 with a time-dependent inertial term. For example, Larionov, Batty, and Bridson [2017] employed the unsteady Stokes equations to simulate viscous fluids more faithfully in a broad class of problems. In contrast, this work focuses on the steady (i.e., time-independent) Stokes equations as a simplified testbed for evaluating the generalizability of WoB. While Eq. 4.15 is sometimes explicitly referred to as the steady-state Stokes equations to distinguish it from its unsteady counterpart, we will refer to it simply as the Stokes equations throughout this thesis.

The fundamental solutions of the Stokes equations describe the response to a point force $\mathbf{f}$ (a Dirac delta distribution) in the momentum equation of Eq. 4.15, within an infinite domain $\Omega = \mathbb{R}^d$, where d is the spatial dimension. Since $\mathbf{f}$ is a d-dimensional vector, the fundamental solutions characterize the influence of each component of the force on each component of the velocity and other related quantities, resulting in *matrix-valued* fundamental solutions. Two fundamental solutions are of particular interest in this context: the *Stokeslet* and the *stresslet*. The Stokeslet represents the velocity field induced by a concentrated point force, while the stresslet represents the corresponding effect on the stress field ($\sigma = -p\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^\top)$). In boundary integral formulations, we are particularly concerned with the stresslet dotted by the outward normal vector (which yields $\sigma \cdot \mathbf{n}$).

In two dimensions, the Stokeslet and the stresslet (dotted by the outward normal) are given by [Abousleiman and Cheng 1994]

$$S(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi\mu r^2}\left(-r^2 \log(r)\mathbf{I} + \mathbf{r}\mathbf{r}^\intercal\right), \quad T(\mathbf{x}, \mathbf{y}) = -\frac{\mathbf{r} \cdot \mathbf{n}_\mathbf{y}}{\pi r^4}\mathbf{r}\mathbf{r}^\intercal, \tag{4.16}$$

respectively, and in three dimensions, they are given by

$$S(\mathbf{x},\mathbf{y}) = \frac{1}{8\pi\mu r^3}\left(r^2\mathbf{I} + \mathbf{r}\mathbf{r}^\intercal\right), \quad T(\mathbf{x},\mathbf{y}) = -\frac{3\mathbf{r}\cdot\mathbf{n}_\mathbf{y}}{4\pi r^5}\mathbf{r}\mathbf{r}^\intercal, \tag{4.17}$$

respectively, where $\mathbf{r} = \mathbf{y} - \mathbf{x}$, $r = \|\mathbf{r}\|_2$, $\mathbf{n}_\mathbf{y}$ is the outward normal at $\mathbf{y} \in \partial\Omega$, and $\mathbf{I}$ is the identity matrix.

Analogously to the Poisson equation, the solution to the Stokes equations in an infinite domain can be expressed as a convolution of the fundamental solution with a given body force distribution. Several prior works [Cortez 2001; Cortez et al. 2005; Sugimoto et al. 2024c] make direct use of this relationship. In contrast, we will utilize these fundamental solutions to define the BIEs and use them in our method.

4.2.2 BIEs for the Stokes Equations

The BIE formulations for the Laplace equation discussed in Section 2.2.3 largely carry over to the Stokes equations, with appropriate substitution of the fundamental solutions. A detailed derivation of such formulations for the Stokes case is provided by Power and Wrobel [1995]. Here, we assume that the body force term $\mathbf{f}$ in Eq. 4.15 is zero and summarize the key resulting equations. Similarly to the Laplace equation, the choice of formulation depends on the type of problem to be solved, and we will have some discussion on this at the end of Section 4.2.3.

The mixed boundary conditions we consider for the Stokes equations (Eq. 4.15) are given by

$$\begin{aligned} \mathbf{u}(\mathbf{x}) &= \overline{\mathbf{u}}_D(\mathbf{x}) \qquad \text{for } \mathbf{x} \in \Gamma_D, \\ \mathbf{q}(\mathbf{x}) &= \overline{\mathbf{q}}_N(\mathbf{x}) \qquad \text{for } \mathbf{x} \in \Gamma_N, \end{aligned} \tag{4.18}$$

where $\mathbf{q}$ denotes the boundary traction, defined as the stress tensor ($\sigma = -p\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^\top)$) dotted with the outward normal ($\mathbf{q} = \sigma \cdot \mathbf{n}$). The two conditions in Eq. 4.18 are commonly referred to as *velocity boundary conditions* and *traction boundary conditions*, respectively, and are also often called Dirichlet and Neumann conditions, in analogy with the Poisson equation. We assume that the boundary of the domain is fully partitioned such that $\partial\Omega = \Gamma = \Gamma_D \cup \Gamma_N$.

Indirect BIE — Single Layer Potential Formulation

The first formulation is the (hydrodynamic) single layer potential-based indirect formulation, which assumes that the velocity field $\mathbf{u}(\mathbf{x})$ inside the domain and on its boundary

can be expressed using a single layer potential:

$$\mathbf{u}(\mathbf{x}) = \int_\Gamma S(\mathbf{x},\mathbf{y})\Psi(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Omega \cup \Gamma, \tag{4.19}$$

where $\Psi(\mathbf{y})$ is a vector-valued unknown boundary density defined over the boundary Γ. From this assumption, we can get an integral equation that involves the traction variable:

$$\Psi(\mathbf{x}) = -\int_\Gamma 2T(\mathbf{y},\mathbf{x})\Psi(\mathbf{y})\,\mathrm{d}\mathbf{y} + 2\mathbf{q}(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \Gamma, \tag{4.20}$$

assuming Γ is sufficiently smooth. This assumption is justified because, in the WoB formulation we use later, we apply the BIE at stochastically sampled points lying on smooth segments of the boundary. On Γ_N, where the traction $\mathbf{q}$ is known, Eq. 4.20 provides a recursive integral equation for Ψ. On Γ_D, where the velocity $\mathbf{u}$ is known, we can derive a different form by multiplying both sides of Eq. 4.19 by a user-defined constant k, moving all terms to one side, and adding $\Psi(\mathbf{x})$ to both sides:

$$\Psi(\mathbf{x}) = \Psi(\mathbf{x}) - k\int_\Gamma S(\mathbf{x},\mathbf{y})\Psi(\mathbf{y})\,\mathrm{d}\mathbf{y} + k\mathbf{u}(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \Gamma. \tag{4.21}$$

This form introduces the unknown Ψ on both sides of the equation, making it amenable to recursive Monte Carlo estimation, as discussed previously for the Laplace equation in Section 3.2.3.

Indirect BIE – Double Layer Potential Formulation

Alternatively, we may assume that the solution takes the form of a (hydrodynamic) double layer potential:

$$\mathbf{u}(\mathbf{x}) = -\int_\Gamma T(\mathbf{x},\mathbf{y})\Phi(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x} \in \Omega, \tag{4.22}$$

where $\Phi(\mathbf{y})$ is another vector-valued unknown boundary density. Taking its limit to the boundary yields

$$\Phi(\mathbf{x}) = \int_\Gamma 2T(\mathbf{x},\mathbf{y})\Phi(\mathbf{y})\,\mathrm{d}\mathbf{y} + 2\mathbf{u}(\mathbf{x}) \quad \text{for} \quad \mathbf{x} \in \Gamma, \tag{4.23}$$

a BIE about the unknown density function, when applied on Γ_D. On Γ_N, we can derive a corresponding equation for Φ in terms of the known traction by taking a derivative of Eq. 4.22 and making $\mathbf{x}$ approach the boundary; however, the corresponding integral

equation has a hypersingular kernel, which introduces significant analytical and numerical challenges as we described in Section 3.4.1. Unless special techniques are employed, it is generally advisable to avoid using this equation in numerical computations due to potential instability and high error; thus, we avoid such a formulation.

Direct BIE Formulation

In contrast to the indirect formulations, the direct BIE formulation expresses the solution directly in terms of the simulation variables $\mathbf{u}$ and $\mathbf{q}$ themselves, rather than in terms of auxiliary densities. Inside the domain, the velocity is given by

$$\mathbf{u}(\mathbf{x}) = -\int_\Gamma T(\mathbf{x},\mathbf{y})\mathbf{u}(\mathbf{y})\,\mathrm{d}\mathbf{y} + \int_\Gamma S(\mathbf{x},\mathbf{y})\mathbf{q}(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x}\in\Omega, \tag{4.24}$$

and on the boundary, it satisfies

$$\mathbf{u}(\mathbf{x}) = -\int_\Gamma 2T(\mathbf{x},\mathbf{y})\mathbf{u}(\mathbf{y})\,\mathrm{d}\mathbf{y} + \int_\Gamma 2S(\mathbf{x},\mathbf{y})\mathbf{q}(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x}\in\Gamma. \tag{4.25}$$

When $\mathbf{x}\in\Gamma_N$, where the traction is known, Eq. 4.25 provides an equation to estimate the unknown velocity $\mathbf{u}$. On Γ_D, where the velocity is known, we apply a transformation analogous to the one used to obtain Eq. 4.21 from Eq. 4.19, yielding

$$\begin{aligned}\mathbf{q}(\mathbf{x}) =& \mathbf{q}(\mathbf{x}) - k\mathbf{u}(\mathbf{x}) \\ &- k\int_\Gamma 2T(\mathbf{x},\mathbf{y})\mathbf{u}(\mathbf{y})\,\mathrm{d}\mathbf{y} + k\int_\Gamma 2S(\mathbf{x},\mathbf{y})\mathbf{q}(\mathbf{y})\,\mathrm{d}\mathbf{y} \quad \text{for} \quad \mathbf{x}\in\Gamma.\end{aligned} \tag{4.26}$$

This form again places the unknown on both sides, enabling recursive estimation.

4.2.3 Walk on Boundary Method for the Stokes Equations

Backward Estimator

Based on the BIEs discussed above, we now design a method to estimate the solution using recursive Monte Carlo estimation. We follow a similar procedure to the one presented for the Laplace equation in Section 3.2.4, now adapted to the Stokes equations. In this section, we describe the single layer potential formulation as a concrete example; analogous derivations apply to the other formulations. From Equations 4.20 and 4.21, we derive

recursive Monte Carlo estimators for the unknown boundary density Ψ, denoted here as $\hat{\Psi}$.

For points $\mathbf{x}_i \in \Gamma_N$, where the traction is known, we apply the integral equation in Eq. 4.20, and estimate $\hat{\Psi}$ recursively as:

$$\hat{\Psi}(\mathbf{x}_i) \coloneqq \frac{-2T(\mathbf{x}_{i+1}, \mathbf{x}_i)}{p(\mathbf{x}_{i+1}|\mathbf{x}_i)} \hat{\Psi}(\mathbf{x}_{i+1}) + 2\mathbf{q}(\mathbf{x}_i), \tag{4.27}$$

where $\mathbf{x}_{i+1} \in \Gamma$ is sampled according to the conditional density $p(\mathbf{x}_{i+1}|\mathbf{x}_i)$. For points $\mathbf{x}_i \in \Gamma_D$, where the velocity is known, we apply Eq. 4.21 and estimate the density using:

$$\hat{\Psi}(\mathbf{x}_i) \coloneqq \begin{cases} -\frac{k}{p_k} \cdot \frac{S(\mathbf{x}_i, \mathbf{x}_{i+1})}{p(\mathbf{x}_{i+1}|\mathbf{x}_i)} \hat{\Psi}(\mathbf{x}_{i+1}) + k\mathbf{u}(\mathbf{x}_i) & \text{with probability } p_k, \\ \frac{1}{1-p_k} \hat{\Psi}(\mathbf{x}_{i+1}) + k\mathbf{u}(\mathbf{x}_i) & \text{with probability } 1 - p_k, \end{cases} \tag{4.28}$$

where again $\mathbf{x}_{i+1}$ is sampled from $p(\mathbf{x}_{i+1}|\mathbf{x}_i)$, except in the second case, where we set $\mathbf{x}_{i+1} \coloneqq \mathbf{x}_i$.

Because the recursive relationships above do not terminate naturally, we introduce a truncation scheme by fixing the recursion depth to a finite number M. At the final recursion step $i = M$, we discard the recursive term and instead assign a constant-weighted value based on the known boundary data:

$$\hat{\Psi}(\mathbf{x}_M) \coloneqq \mathbf{q}(\mathbf{x}_M) \qquad \text{for } \mathbf{x}_M \in \Gamma_N, \tag{4.29}$$

$$\hat{\Psi}(\mathbf{x}_M) \coloneqq (k/2)\mathbf{u}(\mathbf{x}_M) \qquad \text{for } \mathbf{x}_M \in \Gamma_D. \tag{4.30}$$

This truncation strategy introduces a bias to the estimator. However, this particular form of truncation with a constant multiplier of $1/2$ is not the only option. Other alternatives include: series acceleration techniques, which use nonuniform weights to reduce bias, and Russian Roulette path termination, which yields an unbiased estimator at the cost of increased variance. The truncation method used here is motivated by the estimators developed for interior Dirichlet problems of the Laplace equation in Section 3.2.1. While the structure of the recursive estimator is similar, the specific behavior may differ for other boundary conditions or other PDEs like the Stokes equations, due to differences in the integral equation properties. Nevertheless, we empirically observe that this truncation scheme results in finite bias, and that the bias tends to decrease as the truncation depth M increases (Fig. 4.7). We leave the design of more advanced weighting strategies and a formal bias-variance analysis for future work.

Given the estimator for $\hat{\Psi}$, we can now construct an estimator for the velocity $\mathbf{u}$ at a target point $\mathbf{x}_0$ using Eq. 4.19:

$$\hat{\mathbf{u}}(\mathbf{x}_0) = \frac{S(\mathbf{x}_0, \mathbf{x}_1)}{p(\mathbf{x}_1|\mathbf{x}_0)} \hat{\Psi}(\mathbf{x}_1), \tag{4.31}$$

where $\mathbf{x}_1 \in \Gamma$ is sampled from the boundary according to a prescribed probability density function $p(\mathbf{x}_1|\mathbf{x}_0)$. By generating N independent samples of $\hat{\mathbf{u}}(\mathbf{x}_0)$ and averaging them, we obtain a Monte Carlo estimate of the velocity: $\mathbf{u}(\mathbf{x}_0) \approx \frac{1}{N}\sum_{n=1}^{N} \hat{\mathbf{u}}(\mathbf{x}_0)$. This estimator follows a *backward* sampling strategy, analogous to backward path tracing in rendering. That is, the sampling path begins at the query point where we want to estimate the solution, and recursively traces contributions from the boundary. Due to the structural similarity between the BIEs and the rendering equation, alternative estimation strategies for the unknown densities, inspired by rendering techniques, are also possible.

Boundary Value Caching

The naive backward estimator described above can be computationally expensive, as it constructs independent path samples from scratch for each evaluation point in the domain. To mitigate this cost, we adopt a boundary value caching strategy similar to the one discussed in Section 3.3. Specifically, we first sample a set of N_c boundary points $\mathbf{y}_c$ and estimate the unknown density function at each of these points using N recursive Monte Carlo samples: $\Psi(\mathbf{y}_c) \approx \Psi_c := \frac{1}{N}\sum_{n=1}^{N} \hat{\Psi}(\mathbf{y}_c)$. Once these cached boundary values are available, we can approximate the solution at any interior point $\mathbf{x}$ using the cached samples:

$$u(\mathbf{x}) \approx \frac{1}{N_c}\sum_{c=1}^{N_c} \frac{S(\mathbf{x}, \mathbf{y}_c)\Psi_c}{p(\mathbf{y}_c)}. \tag{4.32}$$

This caching approach increases the effective number of contributing paths per evaluation point, thereby improving efficiency, but at the expense of introducing correlations between estimates at different locations. In practice, this correlation leads to spatially smooth estimates, which are particularly desirable in fluid simulation contexts where the Stokes equations are applied. Furthermore, as seen in Eq. 4.32, the solution constructed via caching is globally defined as a weighted sum of fundamental solutions. While this approximation may not strictly satisfy the imposed boundary conditions, it exactly satisfies the momentum equation and the incompressibility constraint in Eq. 4.15, since both are satisfied by each fundamental solution individually and preserved under linear superposition. In particular, the exact satisfaction of the incompressibility condition is a significant advantage in fluid simulation, as it ensures there is no artificial source or sink of fluid, preserving mass conservation. Another key advantage of this strategy is its flexibility. Because the boundary cache points are (typically) sampled independently of any specific evaluation location, the cached values can be reused across multiple evaluations. This allows for efficient solution queries at many or dynamically changing points without requiring the regeneration of full Monte Carlo paths for each query point.

Sampling

Using a well-designed sampling strategy can significantly reduce the variance of the estimator. In Section 3.2.4, we introduced several sampling strategies for the Laplace equation, including ray-intersection sampling, resampled importance sampling (RIS) [Talbot et al. 2005], uniform area-measure sampling, and boundary value importance sampling. For the Stokes equations, these strategies can also be applied, with a few careful modifications.

The fundamental solution $T(\mathbf{x}, \mathbf{y})$ for the Stokes equations evaluates to zero at points that ray-intersection sampling would not sample, similar to the Laplace equation's case. This property enables the use of ray-intersection sampling for pure velocity boundary conditions (via the double layer potential formulation), and for pure traction boundary conditions (using either the single layer potential or the direct formulation), assuming a simply connected domain.

For RIS and boundary value importance sampling, we must define scalar-valued weights as the target distribution. With RIS, for example, we may use the norm of the integral kernel function as the target distribution. Similarly, for boundary value importance sampling, we can sample according to the norm of the vector-valued boundary condition.

In the results we show in Section 4.3.2, we use uniform area-measure sampling to select boundary cache locations. In the recursive estimate of unknown density functions for boundary value caching, we employ RIS with uniform area-measure sampling as the source distribution and the norm of the integral kernel as the target distribution, using 64 candidate samples. The non-recursive part of the direct formulation-based estimator samples boundary points according to the norm of the boundary value.

Choice of BIE Formulation

The choice of BIE for each boundary problem for the Laplace equation applies to the Stokes equations, with the analogy between the boundary conditions (Dirichlet — velocity, and Neumann — traction). In Fig. 4.3, for the simply connected domain problem with velocity boundary conditions (top), both the single layer and the double layer formulations give reasonable results. The direct formulation estimator, however, gives a velocity field with too small a magnitude. This could be due to the poor choice of the constant multiplier k in Eq. 4.26 as the valid range of k should be boundary-shape dependent, and further investigation is required to determine this value automatically. While we have seen a visually correct result, this discussion applies to the single layer potential formulation, too.

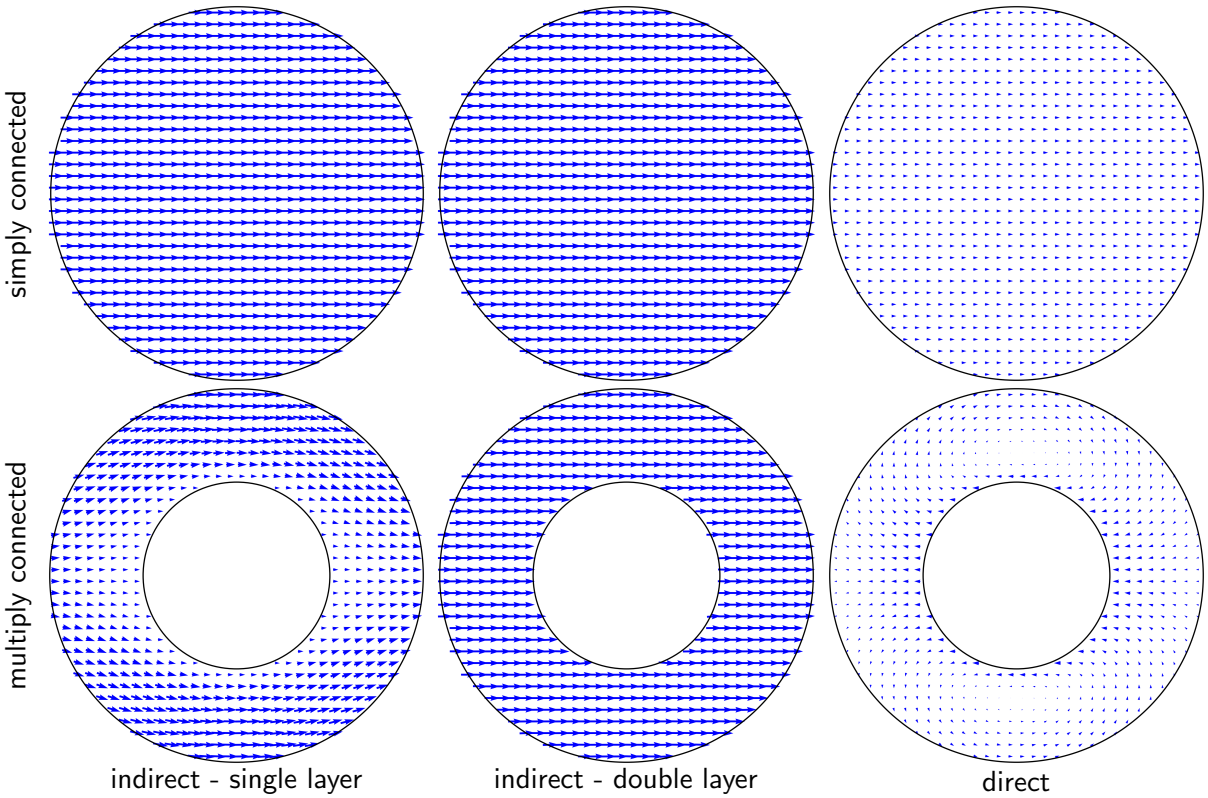


Figure 4.3: Comparison of the velocity field estimates of the three solvers for a simply connected domain problem (top) and a multiply-connected domain problem (bottom) under a velocity boundary condition with Stokes WoB. On the outer circle boundary, we set a velocity boundary condition $\mathbf{u} = (1, 0)^\intercal$ in all cases, and for the multiply connected domain problem, we set a velocity boundary condition $\mathbf{u} = (0, 0)^\intercal$ everywhere on the additional inner circle boundary. The expected solution is a uniform horizontal flow $\mathbf{u} = (1, 0)^\intercal$ for the simply connected case, and a flow that smoothly interpolates from $\mathbf{u} = (1, 0)^\intercal$ to $(0, 0)^\intercal$ for the multiply connected case. Only the single layer potential formulation gives the correct estimates in both types of domains.

In Section 3.2, we focused mainly on simply connected domain problems, and we did not have so much discussion on multiply connected domain problems. In Fig. 4.3, for the multiply connected domain problem with velocity boundary conditions (bottom), only the single layer potential formulation gives a reasonable result. As noted above, the direct formulation fails to estimate the solution correctly. Additionally, we observe that the double layer formulation also does not work in this case. Sabelfeld and Simonov [1994] discusses an extension of the double layer-based estimator to support multiply connected domains by converting the boundary value problem to the one for which the original WoB method works. The current implementation we use for Fig. 4.3 does not implement such a method, and further investigations are required to enable such methods for the Stokes equations.

For the above reasons, we use the estimator with the single layer potential formulation in the results we report in Section 4.3.2. We use $k = 4$ and $p_k = 2/3$ in our experiments.

4.3 Results

We implemented the diffusion and Stokes WoB solvers by extending the CUDA implementation of WoB used in Section 3.2. The only implementation differences are the addition of time tracking for the diffusion equation and support for matrix-valued kernel functions and vector-valued boundary density functions for the Stokes equations.

4.3.1 Diffusion Equation

Convergence

We will compare the numerical result of our diffusion equation solver with analytical solutions to verify the correctness of the method and analyze its error. All solutions we consider assume that the diffusion coefficient $\nu = 1$. We consider four test cases in this study:

- *2D interior domain problem.* In a cubic domain $[-0.5, 0.5]^2$, we set the initial condition to be 1 on the lower half of the domain and 0 on the upper half. With zero Dirichlet conditions, we can get an analytical solution expressed in terms of an infinite series with two nested infinite summations, using Fourier transformations, as shown by Daileda [2012]. We truncate the series after 1000 terms for both summations to get an estimate we use to compare our result against.

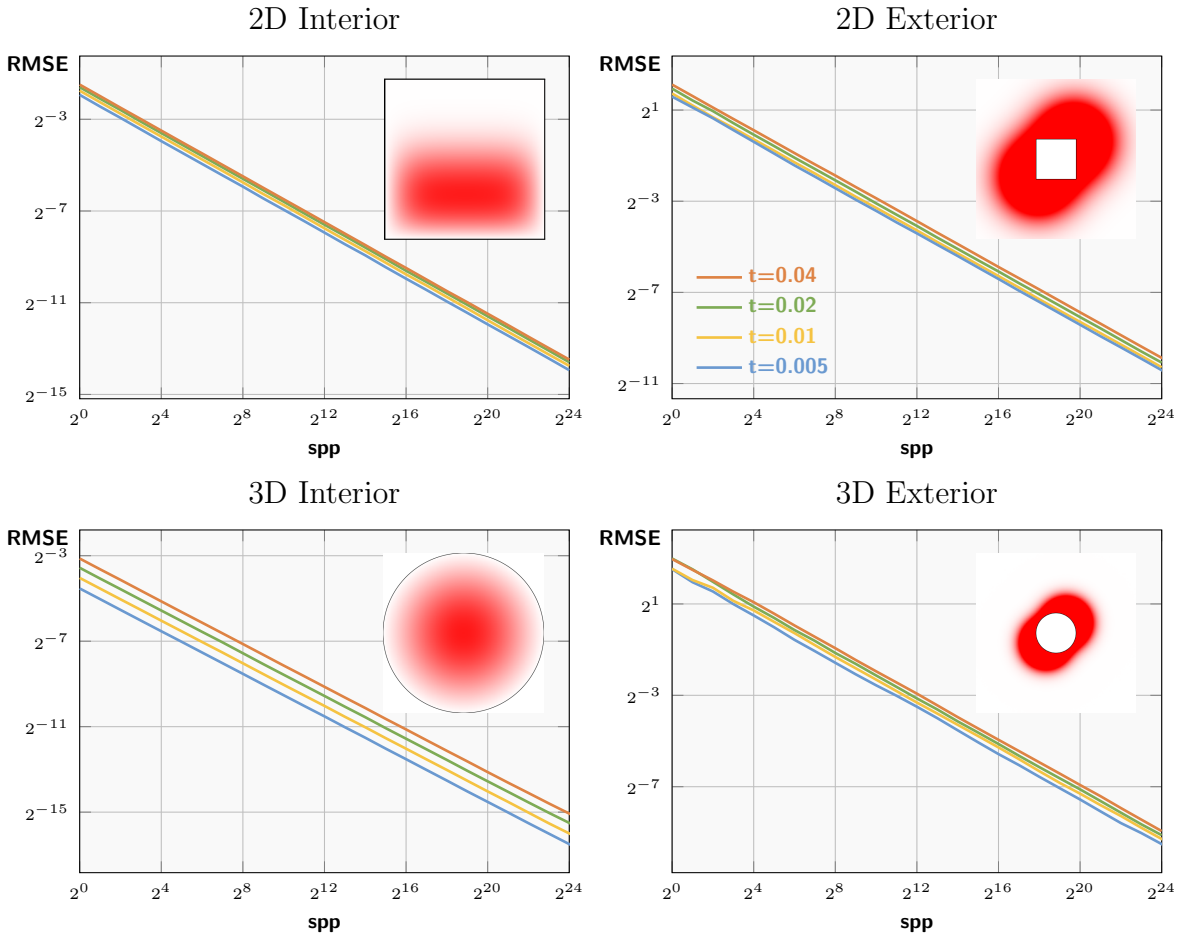


Figure 4.4: Convergence plots for the diffusion equation WoB. For the scenes described in Section 4.3.1, we plot the root mean squared errors (RMSEs) at four different times as the number of samples per point increases. In all cases, we observe the expected Monte Carlo convergence without any clear signs of bias, indicating the unbiasedness of the method. The errors tend to be large for larger t's. For each scene, we also show a visualization of the solution at $t = 0.01$ using 2^{24} samples.

- *2D exterior domain problem.* We first consider the solution to an infinite domain problem, defined by the weighted sum of a few fundamental solutions to the diffusion equation. We can evaluate this solution analytically. If we set the initial condition and the boundary conditions in an exterior domain so that the values would match those given by the infinite domain problem, we get an exterior domain problem with the solution given by the solution to the infinite domain problem. In our study, we set the exterior domain to be $\mathbb{R}^2$ $[-0.25, 0.25]^2$ and the infinite-domain solution to be the sum of two heat kernels, each centered near opposite corners of the domain and weighted equally.

- *3D interior domain problem.* In a unit ball domain, we set the initial condition to be a radially symmetric function $u_0(r) = \frac{\sin(\pi r)}{\pi r}$, and set the zero Dirichlet boundary condition. The analytical solution to this problem is $u(r, t) = \frac{\sin(\pi r)}{\pi r} \exp(-\pi^2 t)$.

- *3D exterior domain problem.* Similarly to the 2D exterior domain problem, we consider a superposition of two heat kernels placed diagonally as the analytical solution. The computational domain is defined as the exterior of a ball of radius 0.25 centered at the origin, i.e., the domain $\mathbb{R}^3 \setminus B(0, 0.25)$.

Fig. 4.4 shows the convergence plots for these test cases. We estimate the solutions at 256^2 grid points at fixed times. For 3D scenes, we extract a 2D slice and evaluate the solution on that slice. In all cases, we observe convergence toward the analytical solutions. Importantly, our method provides pointwise estimates of the solution in both space and time, without requiring time stepping schemes commonly used in discretization-based methods to advance the solution. Notably, all convergence curves appear linear, indicating the *unbiasedness* of our estimator. While small numerical errors and geometric query inaccuracies may introduce minor deviations from exact unbiasedness, the overall behavior remains consistently unbiased across our results.

Runtime and Noise

On a workstation equipped with an NVIDIA RTX A5000 GPU, the experiment for the 3D interior problem at $t = 0.005$ took 9.48 minutes with spp= 2^{24}. In comparison, estimating the solution at $t = 0.010$, 0.020, and 0.040 with the same number of samples per pixel took 10.5, 11.7, and 13.7 minutes, respectively. The runtime at $t = 0.040$ represents an approximate 45% increase over the runtime at $t = 0.005$. This increase is likely due to the greater number of recursion steps required as the evaluation time t increases, possibly compounded by increased execution divergence.

With an equal number of samples, the error also tends to grow with increasing t, as deeper recursion generally leads to higher variance in this setting. This trend is confirmed by the results shown in Fig. 4.5. Our method naturally supports time-dependent boundary conditions without requiring time discretization. However, as the query time t increases, the associated variance grows. One possible future direction is to introduce a cache at an intermediate time to reduce computational cost, potentially at the expense of introducing some bias due to the cached approximation.

Additionally, as the domain geometry becomes more complex, the variance increases, and the method requires longer execution times to achieve results with comparable noise levels. Similar to the case of the Laplace equation, this is partly due to the increased complexity of the geometric queries (i.e., ray intersection sampling) and the greater variance introduced by the stochastic selection of paths. Additionally, for the diffusion equations in particular, small geometric features tend to require more recursive steps for a path to reach the initial time. This is because the sampled time decrement is proportional to the squared distance between two consecutive spatial points in our sampling strategy. The results shown in Fig. 4.1 were obtained in 13 seconds, 24 seconds, 100 minutes, and 100 minutes, respectively, from left to right.

Path Truncation

While the diffusion WoB solver offers the attractive property of unbiased estimation, its variance can become quite large for domains with complex boundaries, as noted above. In particular, for non-convex domains, increased recursion depth can lead to a blowup in variance, since the stochastic selection of intersection points may cause exponential growth in the number of paths. To mitigate this issue, in the results shown in Fig. 4.1, we truncate paths after eight recursion steps. This truncation introduces bias, but the proportion of truncated paths remained low in these experiments, suggesting minimal additional bias. We expect this bias to increase when the query time t is larger. For the convergence study (Fig. 4.4), we disabled path truncation. Fig. 4.6 compares results with and without truncation. As future work, one could consider modifying the sampling strategy to avoid this issue or designing an automatic path termination strategy to better balance the variance-bias tradeoff. Moreover, rather than discarding contributions from estimates beyond a fixed recursion depth, introducing Russian roulette path termination could offer a better variance–bias tradeoff.

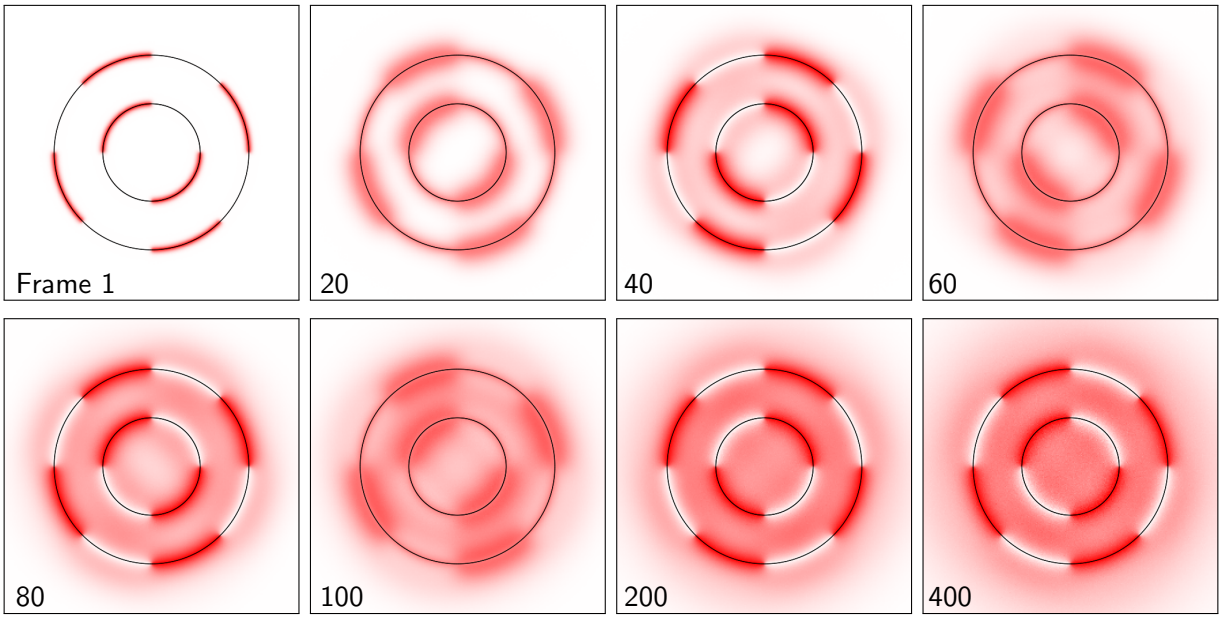


Figure 4.5: Diffusion equation WoB with time-dependent boundary conditions. In this simulation, we gradually and periodically change the Dirichlet boundary values over time. The method correctly handles such boundary conditions, with the solution at the center eventually approaching the average of the boundary values. We also observe that the noise increases as time progresses, especially the farther it is from the initial time.

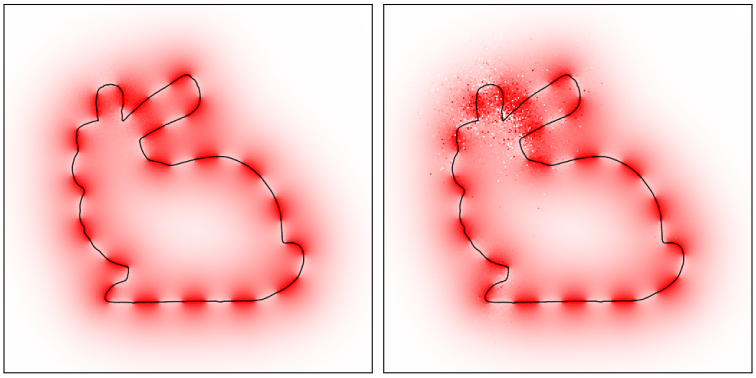


Figure 4.6: Diffusion equation WoB with (left) and without (right) path truncation with the same sample path count. For this scene, path truncation can effectively reduce noise in the result, as we observe around the concavity near the ears of this bunny shape. The estimate becomes biased, though not noticeable in this experiment with a relatively small t.

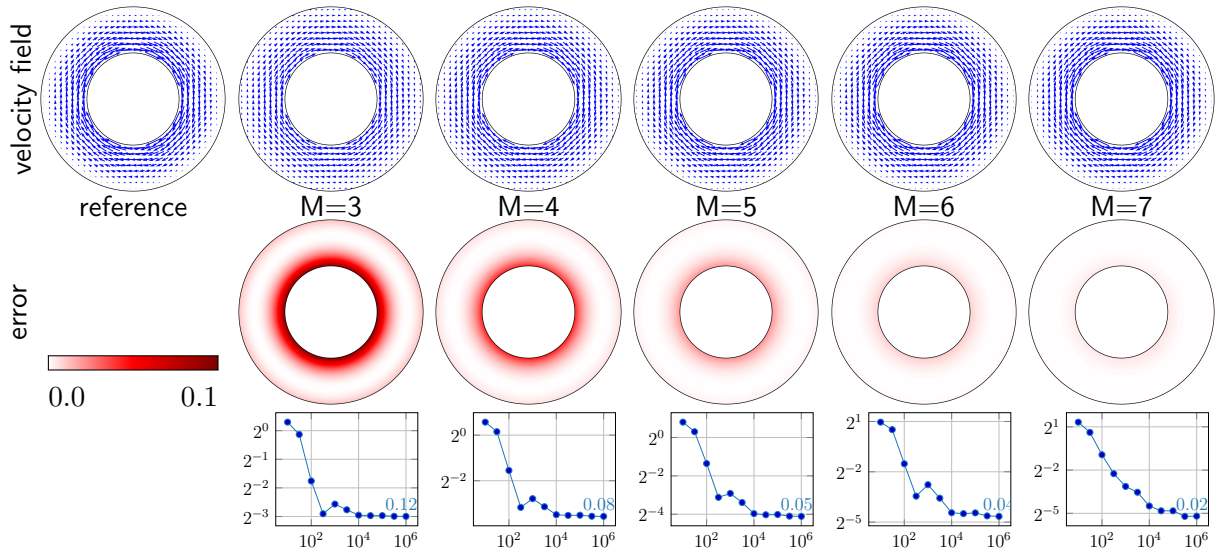


Figure 4.7: Path truncation and bias with the Stokes equation WoB. We show the velocity field estimate (top), error visualization (middle), and error curve (bottom) for the 2D velocity boundary condition estimators with different path lengths M (left to right). The error visualization shows the absolute error measured against the analytical solution (left), and the error plot shows the decay of the root mean squared error (vertical axis) with respect to the number of boundary cache points N_c (horizontal axis). The number displayed in each error plot shows the error with $N_c = 10^6$. The bias decreases as path length increases, but more samples are needed to achieve convergence.

4.3.2 Stokes Equations

The test scene configuration we consider here is the *Taylor-Couette flow*, the flow of a viscous liquid bounded by a slowly rotating inner cylinder and a stationary outer cylinder. We observe an interesting time-reversible phenomenon in this setup in the real footage (Fig. 4.2, top), although some diffusion effects remain visible due to the inertial terms in the Navier-Stokes equations not being completely negligible in reality. In the limit of negligible inertial effects, the analytical flow field is known: in 2D, when the height direction can be ignored, and in 3D, when the domain is considered infinitely long in the vertical direction. In those cases, the analytical angular velocity field is given by

$$\omega(\mathbf{r}) = \omega_1 \frac{\frac{R_2}{r} - \frac{r}{R_2}}{\frac{R_2}{R_1} - \frac{R_1}{R_2}}, \tag{4.33}$$

where the centers of the cylinders are at the origin, R_1 is the radius of the inner cylinder, R_2 is the radius of the outer cylinder, ω_1 is the angular velocity of the inner cylinder, and $r = \|\mathbf{r}\|_2$. In the results we present, we use a dynamic viscosity of $\mu = 1$. We set the angular velocity so the velocity magnitude of the inner cylinder boundary is 1, and impose zero velocity on the outer cylinder; these are the velocity boundary conditions we enforce. In 2D, this setup provides a complete set of boundary conditions, with velocity specified everywhere, i.e., $\Gamma_D = \Gamma$. In 3D, we additionally model the liquid-air interface Γ_N with a zero-traction (free-surface) boundary condition, making the problem a mixed boundary problem.

Path Truncation and Bias

In Fig. 4.7, we show the results of WoB for the Stokes equations with various path lengths. Similar to the case of the Laplace equation, increasing the path length M reduces the bias, at the cost of increased variance. We arrive at this conclusion by observing that the error curves flatten out as the sample path count N_c increases.

Number of Boundary Cache Points and Error

In Fig. 4.8, we test the method with three different boundary cache point counts N_c. As expected, we observe smaller errors with more cache points. In this example, we use a constant path length ($M = 7$) and a constant number of samples per cache point ($N = 10^5$). We expect that increasing the number of samples per cache point would also

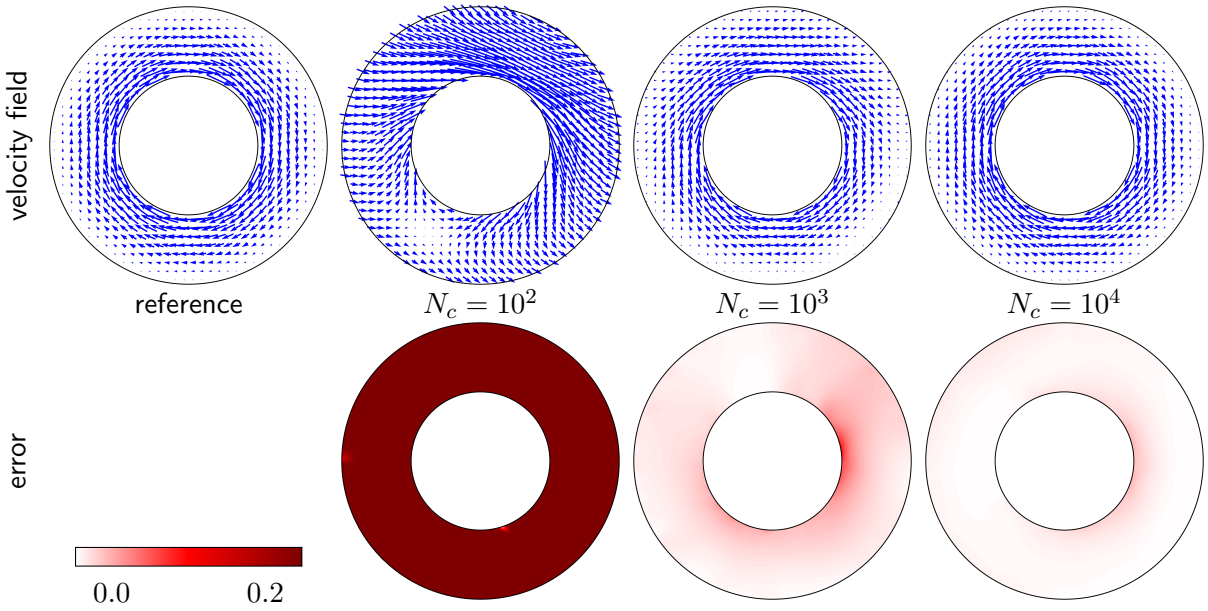


Figure 4.8: We estimate the velocity field (top) and show the error (bottom) against the analytical solution (left) with different numbers of boundary cache points. As expected, having more cache points results in a lower error.

reduce the error; however, if too few boundary cache points are used, the error eventually stops decreasing. On the other hand, using too many cache points increases memory usage and incurs higher computational cost when reconstructing the solution at a point from the cache. A more detailed study of this tradeoff is left for future work.

2D WoB Flow Estimate

In Fig. 4.9, we show frames from an animation generated with WoB. In this example, we solve for the velocity field from the boundary values with $M = 7$, $N_c = 10^6$, and $N = 10^5$, and advect marker particles using the fourth-order Runge-Kutta method. During advection, we evaluate the velocity at the exact particle locations using the boundary cache. This discretization-free evaluation is a unique feature of Monte Carlo-based estimators. The simulation accurately returns the particles to their original positions after the inner cylinder is rotated in both clockwise and counterclockwise directions. Our method does not suffer from traditional discretization or interpolation artifacts.

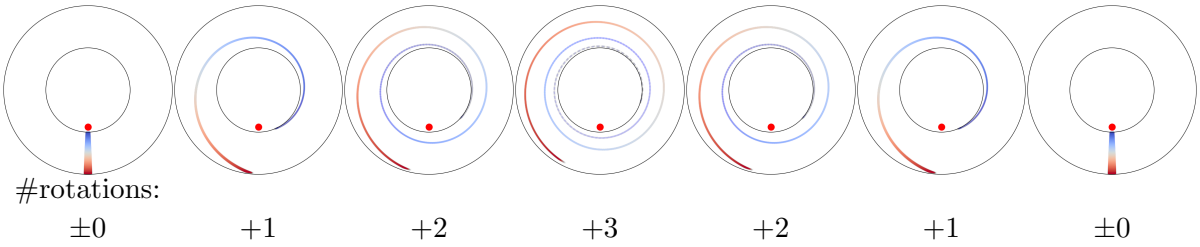


Figure 4.9: Result with the 2D velocity boundary condition. We rotate the inner cylinder in the clockwise direction three times and rotate it back. The numbers listed at the bottom indicate how many times the inner cylinder rotates in the clockwise direction. The advected marker particles go back to their original positions after the simulation, as expected.

3D Flow with Mixed Boundaries

In Fig. 4.2, we simulate the 3D Taylor-Couette flow scene similar to the 2D example in Fig. 4.9, and we similarly observe a reasonable result. In addition to the difference in dimensions, we have an additional free surface where the traction value is specified instead of the velocity value. Thus, the WoB estimator solves a mixed boundary problem in this case. The parameters we use here are $M = 7$, $N_c = 10^6$, and $N = 10^5$.

Chapter 5

Monte Carlo PDE Solvers for Dynamic and Nonlinear Equations — Navier–Stokes Equations

Equipped with the Monte Carlo PDE solvers for various linear equations discussed thus far, we address the challenging problem of solving the incompressible Navier–Stokes equations, a nonlinear, time-dependent PDE that governs fluid motion. Our Monte Carlo fluid solvers offer several advantages similar to the ones we have discussed so far: flexibility in handling complex geometric boundaries, robustness to noise in input geometry, support for pointwise solution estimation, and straightforward parallelization (Fig. 5.1).

We explore two variants of our approach: one based on a vorticity formulation, and the other on a velocity formulation. The vorticity-based method is expected to better conserve vorticity throughout the simulation, while the velocity-based variant can leverage recent advances in traditional discretization-based fluid solvers in graphics.

In both cases, we apply operator splitting [Park and Kim 2005; Stam 1999] to solve the Navier–Stokes equations, and design pointwise estimators for each substep. With operator splitting, we can separate the nonlinear advection term. The core idea is to reformulate the rest of the substeps as integration problems, which enables us to apply Monte Carlo techniques. When no solid obstacles are present, the integrals are non-recursive, and we can estimate them using simple Monte Carlo integration. In the presence of solid boundaries, we incorporate their effects using WoS or WoB methods. Our methods thus do not require the boundaries of solid objects to be explicitly discretized; for example, there is no need for a cut-cell method [Batty et al. 2007] or a conforming mesh [Feldman et al. 2005] as in

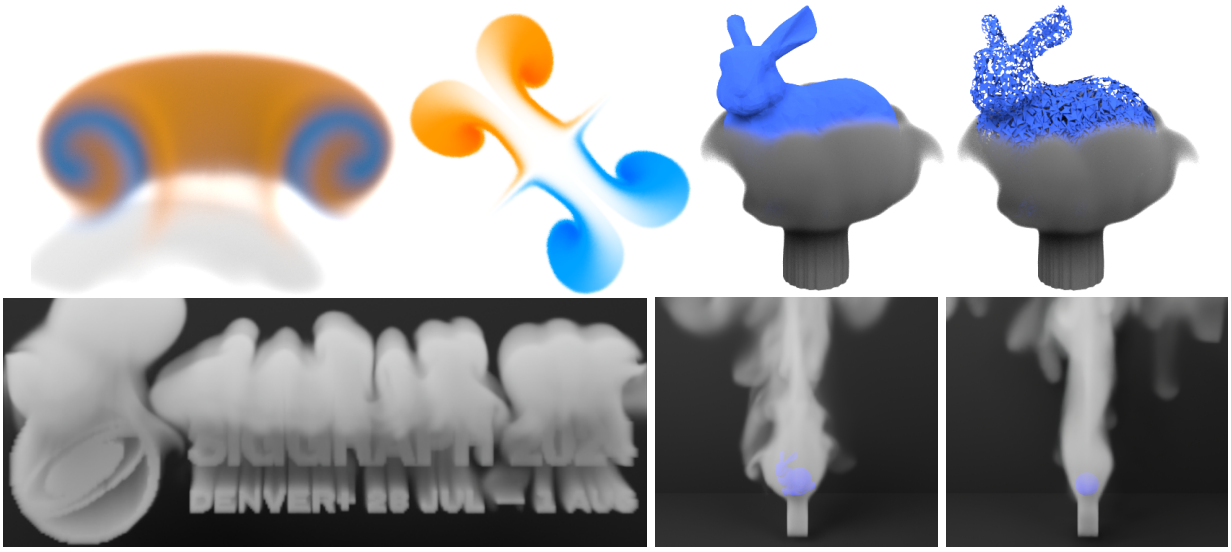


Figure 5.1: Our novel Monte Carlo fluid simulators support both 2D and full 3D simulations of viscous incompressible flows by computing pointwise stochastic solutions to the Navier–Stokes equations, using either a vorticity-based (top) or velocity-based (bottom) formulation. They can treat diverse fluid effects, such as leapfrogging vortex rings (top, left) and colliding jets (top, middle). The adoption of a Monte Carlo method to handle boundaries is well-suited to treating nontrivial boundary geometry (top, right). The velocity-based variant simulates a smoke plume rising from a complex-shaped source (bottom, left) and past a sphere-shaped or bunny-shaped obstacle (bottom, right), where the motion is driven solely by buoyancy forces.

traditional solvers, and the only requirement on the geometry is that it supports simple geometric queries (closest-point or ray-intersection queries).

Constructing pointwise solvers for each substep allows us to define a recursive fluid flow algorithm with striking similarities to simulation and sampling strategies employed in Monte Carlo rendering. This recursive formulation enables purely pointwise estimation of the solution, eliminating the need for spatial discretization. However, in its simplest form, our recursive formulation exhibits exponential computational complexity, akin to how distributed ray tracing scales with the number of light bounces. We overcame this issue with practical caching strategies, such as storing results in a uniform grid.

For the vorticity-based method, we show that our caching scheme also enables us to develop and apply Monte Carlo variance reduction techniques, including importance sampling of the vorticity field and a control variate approach that utilizes the vorticity and velocity fields from the previous time step, thereby increasing sample efficiency and accelerating stochastic estimates.

We validate our methods against standard grid- and particle-based solvers, exploring and summarizing their behavior under different boundary conditions and parameter settings. The pointwise nature of our method allows an easy parallelization of computation. Our work represents the first foray into the space of Monte Carlo methods applied to fluid simulation in graphics. In addition to highlighting the current limitations of our proof-of-concept simulators, we outline and discuss a series of open problems associated with scaling Monte Carlo-based fluid simulators to larger and more challenging flows.

5.1 Background: Fluid Simulation in Graphics

The numerical prediction of fluid motion plays a crucial role in many fields, from weather forecasting to aerospace engineering. In computer graphics, there is a strong demand for realistic animation of flowing water, swirling smoke, flickering flames, and so on. Most approaches address this challenge by numerically solving the classical Navier–Stokes equations under certain assumptions, such as incompressibility or inviscid flow. Our methods aim to numerically solve for a velocity field that satisfies the incompressible Navier–Stokes equations with constant density and viscosity:

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} &= -(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{f}, \\ \nabla\cdot\mathbf{u} &= 0, \end{aligned} \tag{5.1}$$

where $\mathbf{u}$ is velocity, p is pressure, t is time, $\mathbf{f}$ is acceleration due to external forces such as gravity, ν is kinematic viscosity, and ρ is density. The first equation in Eq. 5.1 is the momentum equation, and the second enforces the incompressibility constraint. Wang et al. [2024] provide a comprehensive survey of physics-based fluid simulation techniques in graphics. While the methods we discuss in this chapter fall into this category, it is also important to acknowledge the role of models that introduce aggressive simplifications to the Navier–Stokes equations, often reducing the problem from three dimensions to two. A representative example is the ocean wave simulation technique by Tessendorf [2004], which is widely used in video games and film production.

5.1.1 Spatial Discretizations

To solve the Navier–Stokes equations, we typically represent the field of interest (e.g., velocity or vorticity) using some discrete structure, and evolve it with time-stepping. *Eulerian methods* assume a static mesh or grid through which the fluid flows, and discretize the governing equations using finite difference, finite volume, or finite element techniques [Bridson 2015; Fedkiw et al. 2001; Foster and Fedkiw 2001; Foster and Metaxas 1996; Stam 1999]. *Lagrangian methods*, in contrast, use degrees of freedom that move with the fluid. These are often implemented using mesh-free particles, as in smoothed particle hydrodynamics [Desbrun and Gascuel 1996; Koschier et al. 2022; Müller et al. 2003], or using time-evolving meshes that must be adapted when they become overly distorted [Clausen et al. 2013; Misztal et al. 2014]. *Hybrid methods*, which combine Lagrangian particles with Eulerian grids, have become popular for offering both the convenience and efficiency of grids and the accurate advection of particles [Brackbill and Ruppel 1986; Harlow 1962; Jiang et al. 2015; Zhu and Bridson 2005]. The Monte Carlo methods we present are *not* strictly Eulerian or Lagrangian, as they can avoid discretizing intermediate fields altogether, at least in theory. However, to reduce computational cost, it is often advantageous to incorporate some form of discrete structure, and we use such practical variants of the methods to discuss numerical results.

5.1.2 Simulation Variables

The methods we have referenced to illustrate different spatial discretization strategies all employ *velocity* and *pressure* as their primary simulation variables. This choice is natural, as the original Navier–Stokes equations are expressed in terms of these quantities. Accordingly, such methods are often referred to as *velocity-based* in the graphics community, while

adjacent fields typically refer to them as the velocity–pressure formulation. This velocity-based formulation remains the most widely used approach in graphics today. However, using alternative simulation variables can offer advantages, such as simplifying method design or improving conservation properties.

A prominent example is the class of *vorticity-based* methods, which, like velocity-based methods, can be implemented in either Eulerian or Lagrangian form. In the Eulerian setting, Elcott et al. [2007] and Mullen et al. [2009] proposed circulation- and energy-preserving schemes, respectively, based on discrete differential geometry. In the Lagrangian setting, vorticity variables lead to vortex particle methods, first introduced by Chorin [1973]. These methods represent vorticity using particles that act as moving point sources, while the corresponding velocities are recovered via the Biot–Savart law (Eq. 5.3). The textbook by Cottet and Koumoutsakos [2000] provides a thorough introduction. In computer graphics, early adaptations of Lagrangian vortex methods employed vortex particles [Park and Kim 2005; Selle et al. 2005] and vortex filaments [Angelidis and Neyret 2005], followed by later work using vortex sheets [Brochu et al. 2012; Pfaff et al. 2012] and vortex segment clouds [Xiong et al. 2021]. The computational cost of naive vortex methods is $O(N^2)$ in the number of vortex elements, motivating acceleration techniques such as the vortex-in-cell method [Couët et al. 1981], the fast multipole method [Greengard and Rokhlin 1987], and PPPM [Zhang and Bridson 2014]. Stream functions in 2D and vector potentials in 3D are closely related to vorticity, as they define velocity as the curl of a potential, ensuring incompressibility by construction. Bridson et al. [2007] used stream functions for procedural flow design, while Ando et al. [2015] used vector potentials to simulate liquids with incompressible bubbles. More broadly, stream function–vorticity formulations have a long history in computational fluid dynamics [Campion-Renson and Crochet 1978; Peeters et al. 1987].

Beyond velocity and vorticity, recent research in graphics has explored alternative formulations using Clebsch variables, impulse variables, and others [Chern et al. 2016; Feng et al. 2023; Nabizadeh et al. 2022, 2024; Xiong et al. 2022; Yang et al. 2021]. In this chapter, we present two Monte Carlo methods for the Navier–Stokes equations: one based on a vorticity-based formulation, and the other on a velocity-based formulation. Although extending Monte Carlo techniques to other variable choices should be feasible, we leave such directions for future exploration.

Probabilistic fluid models. There has been some research into probabilistic treatments of the Navier–Stokes equations, including the Fourier transformation method [Jan and Sznitman 1997], the Lagrangian flow method [Constantin and Iyer 2008], and the

FBSDS approach [Busnello et al. 2005; Delbaen et al. 2015], to name a few. For a more comprehensive review, we refer the reader to the work of Cruzeiro [2020]. However, this body of work has primarily focused on the mathematical properties of the Navier–Stokes equations, such as questions of existence, uniqueness, and regularity; no practical numerical simulation has been conducted to the best of our knowledge.

We can interpret our vorticity-based Monte Carlo solver as a numerical method based on these mathematical analyses—most directly, that of Busnello et al. [2005]—by constructing a stochastic algorithm for the Navier–Stokes equations that explicitly handles viscosity and vortex stretching via the Feynman–Kac formula. Sawhney and Seyb et al. [2022] also used this formula to transform time-independent PDEs with spatially varying coefficients into constant-coefficient problems. In contrast, we apply the formula directly to the time-dependent Navier–Stokes equations. Combined with the Euler–Maruyama integration [Maruyama 1955], this perspective underlies the method described in our published paper [Rioux-Lavoie and Sugimoto et al. 2022], which forms the basis of the vorticity-based method presented in this chapter. In this thesis, we instead present both the vorticity-based and velocity-based Monte Carlo methods under a unified operator splitting perspective, which may be more familiar to a graphics-oriented audience. The vorticity-based method we describe here is identical in algorithm to the one we presented in the paper, but differs in its derivation.

Sabelfeld and Simonov [1994, Section 7.2] discuss WoB solvers for the Navier–Stokes equations, but their method applies only in limited scenarios (two dimensions, no-slip boundaries, short times) and focuses primarily on the mathematical construction of the algorithm, without any numerical validation. In contrast, we support both two- and three-dimensional simulations with no-slip boundary conditions and present a variety of long-time simulation results—key concerns for the graphics community. Furthermore, we develop several variance-reduction strategies and adapt techniques from graphics that are well-suited to such scenarios.

In a quite different vein, there has been a recent resurgence of effort in graphics on Lattice Boltzmann methods within graphics [Li et al. 2018, 2020], including recent advances in handling complex geometries [Lyu et al. 2021]. These methods evolve probability density functions over a discrete lattice, effectively simulating fluid behavior as a kind of cellular automaton. While both approaches employ stochastic elements, our method is conceptually and algorithmically distinct from Lattice Boltzmann methods.

Concurrently with this thesis work, there has been growing interest in applying Monte Carlo methods to related fluid animation problems in the graphics community. Jain et al. [2024] solve the Navier–Stokes equations in the velocity–pressure formulation using the

Walk on Stars Monte Carlo method for the projection step, representing the intermediate velocity field with a neural network. Ye et al. [2024] apply a Monte Carlo approach to the Biot–Savart summation that arises when restoring lost vorticity [Zhang et al. 2015] in a (velocity-based) smoothed particle hydrodynamics simulation.

5.2 Vorticity-Based Monte Carlo Method

Fluid motion is commonly modeled by the Navier-Stokes equations describing the evolution of the velocity field $\mathbf{u}$ in time. By considering the vorticity field $\omega \coloneqq \nabla \times \mathbf{u}$, one can instead derive a time-evolution equation for vorticity. By taking the curl of the incompressible Navier–Stokes equations (Eq. 5.1), we get the well-established vorticity transport equation [Cottet and Koumoutsakos 2000] for incompressible flow in three dimensions:

$$\frac{D_{\mathbf{u}}\omega}{Dt} = (\omega \cdot \nabla)\mathbf{u} + \nu\nabla^2\omega + \nabla \times \mathbf{f}, \tag{5.2}$$

where $\frac{D_{\mathbf{u}}}{Dt} = \frac{\partial}{\partial t} + (\mathbf{u} \cdot \nabla)$ is the material derivative. We can ignore the term associated with acceleration due to external forces, $\nabla \times \mathbf{f}$, if the curl of the acceleration field is zero. For instance, the curl of gravitational acceleration is zero. Thus, we drop this term in our method and leave the treatment of more complex forcing terms to future work. The Navier–Stokes equations, in their velocity form, couple the velocity and the pressure variables in an intricate way: the pressure serves as a Lagrange multiplier that makes the velocity field incompressible. In contrast, this vorticity transport equation offers a straightforward application of the Monte Carlo method.

In the case of 2D problems, the absence of a third dimension reduces the vorticity vector field to a scalar field $\omega = \frac{\partial}{\partial x}\mathbf{u}_y - \frac{\partial}{\partial y}\mathbf{u}_x$. In what follows, we denote the dependence of field quantities on position $\mathbf{x}$ and time t with parentheses, e.g., $\omega(\mathbf{x}, t)$. and we use the following notational conventions in 2D: $\nabla \times \Psi = [\partial_y\Psi, -\partial_x\Psi]$, $\Psi \times \mathbf{A} = [-\Psi A_y, \Psi A_x] = -\mathbf{A} \times \Psi$ and $\nabla \times \mathbf{A} = \partial_x A_y - \partial_y A_x$, where Ψ is a scalar field and $\mathbf{A} = [A_x, A_y]$ is a 2D vector field. For 3D problems, we can interpret these operations in the standard sense.

Operator Splitting with pointwise estimators The vorticity transport equation (Eq. 5.2) is a nonlinear PDE with many terms involved. We solve it by taking discrete time steps with step size Δt. To simplify the problem, as an approximation, we employ an operator splitting technique [Park and Kim 2005], in which we individually consider each component of the equation and solve it sequentially. Concretely, the steps we consider are

1. velocity estimation from vorticity,
2. vortex stretching: $\frac{\partial \omega}{\partial t} = (\omega \cdot \nabla)\mathbf{u}$ (3D only),
3. diffusion: $\frac{\partial \omega}{\partial t} = \nu \nabla^2 \omega$, and
4. advection: $\frac{D_{\mathbf{u}}\omega}{Dt} = 0$.

In contrast to traditional discretization-based solvers, we develop a pointwise (Monte Carlo) estimator for each substep. Specifically, we design a solver for each substep that can estimate the quantity of interest—velocity in the first substep and vorticity in the subsequent ones—at any spatial location, assuming we can similarly query the required quantities from the previous steps at arbitrary spatial points.

This pointwise formulation provides unique advantages. By recursively calling the substep solvers backward in time, toward the initial time where the initial conditions are known, we can estimate the solution of the fluid equations at a specific space-time point of interest without introducing any spatial discretizations and the associated errors (Section 5.4). However, this approach comes with a high computational cost. To mitigate this cost, we can adopt discrete structures to cache the intermediate field values (Section 5.5); still, our method remains agnostic to the underlying discretization of the field due to its pointwise nature.

Below, we formulate a pointwise estimate of the solution to each substep. For notational simplicity, we will drop the time parameter and focus on the update to the vorticity field value (or the velocity field value in substep 1) at each substep. Later, after discussing the substep solvers for both the vorticity-based and velocity-based formulations, we will discuss their theoretical advantages in detail in Section 5.4, and we will describe how we employ these estimators in a practical scenario with, e.g., a uniform grid, in Section 5.5.

5.2.1 Velocity Estimation

Without Prescribed Boundaries

When no boundaries are prescribed in the simulation, we assume problems within an unbounded domain or a periodic domain. For such problems, given the vorticity field $\omega(\mathbf{x})$, we can compute the corresponding velocity field using the *Biot–Savart law* [Cottet and Koumoutsakos 2000]:

$$\mathbf{u}(\mathbf{x}) = \int_{\mathbb{R}^d} \omega(\mathbf{y}) \times K(\mathbf{x} - \mathbf{y}) \, \mathrm{d}\mathbf{y} \tag{5.3}$$

where the Biot–Savart kernel function K is the gradient of a fundamental solution of the Laplace equation; $K(\mathbf{x}-\mathbf{y}) = (\mathbf{x}-\mathbf{y})/(2\pi|\mathbf{x}-\mathbf{y}|^2)$ in 2D and $K(\mathbf{x}-\mathbf{y}) = (\mathbf{x}-\mathbf{y})/(4\pi|\mathbf{x}-\mathbf{y}|^3)$ in 3D. The velocity field obtained from Eq. 5.3 satisfies both the divergence-free property ($\nabla \cdot \mathbf{u} = 0$) and the definition of vorticity ($\omega = \nabla \times \mathbf{u}$).

Monte Carlo estimation Closed-form solutions to Eq. 5.3 are usually unavailable, and we choose to estimate it with Monte Carlo integration. Let $\{\mathbf{y}_i\}_{i=1}^{n_{\text{mc}}}$ be n_{mc} independent samples drawn from probability distribution $p(\mathbf{y} \,|\, \mathbf{x})$. The Monte Carlo estimator of Eq. 5.3 is

$$\langle \mathbf{u}(\mathbf{x}) \rangle = \frac{1}{n_{\text{mc}}} \sum_{i=1}^{n_{\text{mc}}} \frac{\omega(\mathbf{y}_i) \times K(\mathbf{x}-\mathbf{y}_i)}{p(\mathbf{y}_i \,|\, \mathbf{x})} \,. \tag{5.4}$$

To get a Monte Carlo estimate with Eq. 5.4, notice that we can get the estimate at a desired point independently, and we only require the ability to evaluate the vorticity at the position of sample points. Thus, this satisfies our requirement for pointwise evaluation.

With Prescribed Boundaries

To handle free-slip (or no through) solid boundary conditions, we employ a Poisson equation (with Dirichlet boundary conditions) to convert vorticity into a *stream function*, the curl of which yields the desired velocity.

Free-slip boundaries We present our derivations in the 2D setting for clarity, but our ideas generalize fairly naturally to 3D using vector potentials in place of stream functions [Bridson et al. 2007]. We initially consider boundaries with zero normal motion, i.e., $\mathbf{u}_b \cdot \mathbf{n} = 0$; under free-slip conditions, solid tangential velocity has no effect on the flow, so we need not constrain it. Here, we define the stream function $\Psi(\mathbf{x})$ such that

$$\mathbf{u} = -\nabla \times \Psi. \tag{5.5}$$

Note, for convenience, we adopt a flipped sign convention for the stream function, in contrast to the more common definition ($\mathbf{u} = \nabla \times \Psi$ [Bridson et al. 2007]). Combining this expression with the definition of vorticity gives the following Poisson equation (in 2D):

$$\omega = \nabla \times \mathbf{u} = \nabla \times (-\nabla \times \Psi) = \nabla^2 \Psi. \tag{5.6}$$

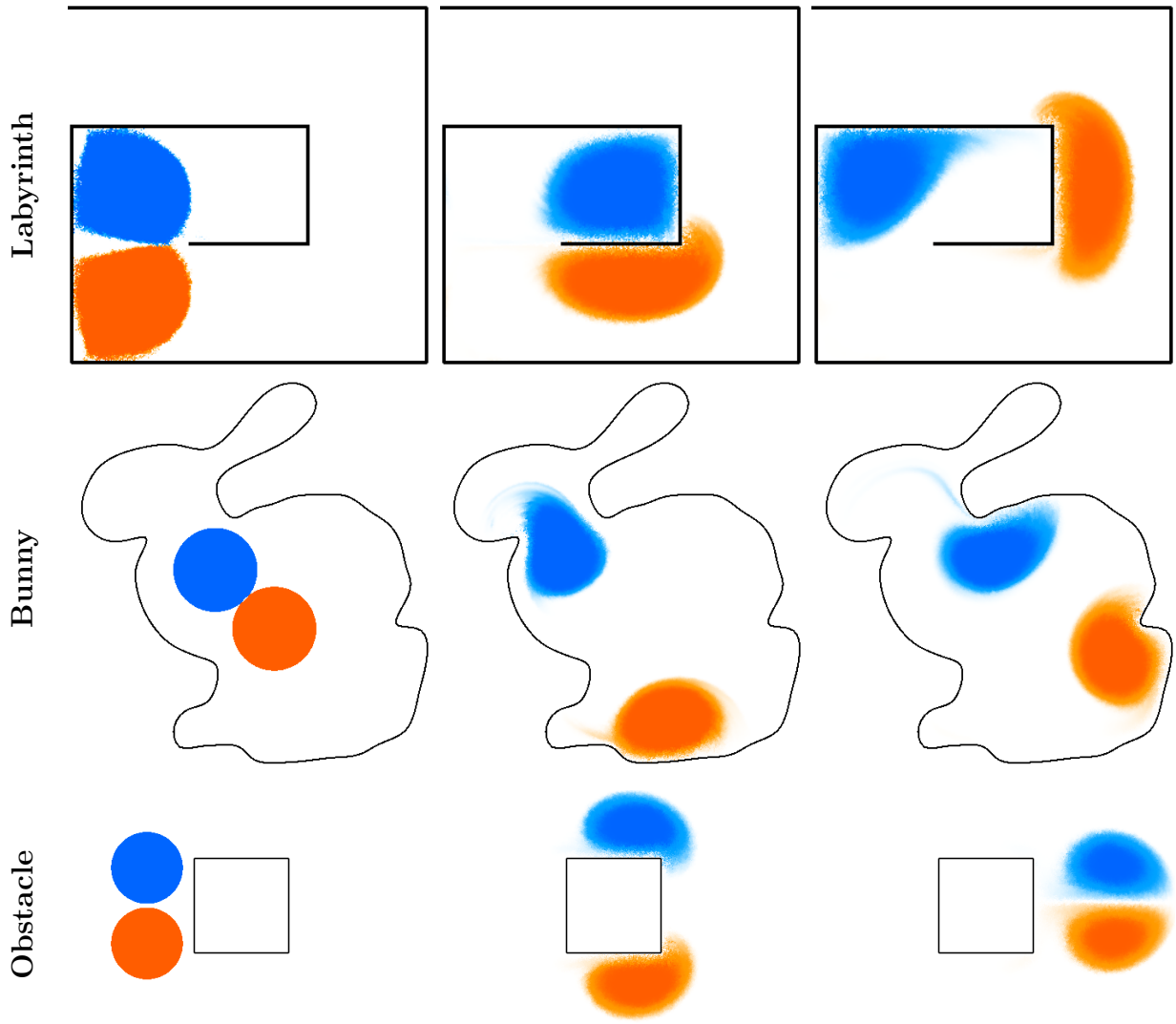


Figure 5.2: Free-slip boundary conditions. Left to right, $t = 0, 5, 10$ seconds. Two vortices propagating in a maze (top), a flow inside a more complex bunny-shaped boundary defined using a signed distance function (middle), and vortices flowing around a solid obstacle in a periodic domain (bottom).

From Eq. 5.5, if Ψ is constant along the boundary (i.e., an isosurface of Ψ), the normal component of the velocity will be zero (i.e., $\mathbf{u} \cdot \mathbf{n} = 0$). Moreover, if the boundary is connected and piecewise smooth, we can arbitrarily set the isovalue to 0 without loss of generality (Fig. 5.2). With multiple disconnected boundaries, we can still set the same isovalue to each of them if we can expect zero net flow between the boundaries. Thus, after solving the Poisson equation for such a stream function, we can obtain a velocity field that satisfies our desired boundary condition. In 3D, the true boundary condition $(\nabla \times \Psi) \cdot \mathbf{n}$ is more involved as it features derivative interactions among multiple components of the vector Ψ [Ando et al. 2015]. Like Bridson et al. [2007], our current 3D results simply use $\Psi = \mathbf{0}$; further study will be needed to properly resolve scenarios with multiple objects and nontrivial topologies.

In summary, given a domain Ω, to recover the velocity field from a vorticity field, while imposing the slip boundary conditions, we use a stream function Ψ of the velocity field $\mathbf{u} = -\nabla \times \Psi$ as an intermediate variable. We solve the following Poisson equation with Dirichlet boundary conditions first, for the gradient of the stream function:

$$\begin{aligned} \nabla^2 \Psi(\mathbf{x}) &= \omega(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Omega, \text{ and} \\ \Psi(\mathbf{y}) &= g(\mathbf{y}) \quad \text{for } \mathbf{y} \in \partial\Omega, \end{aligned} \tag{5.7}$$

where $g(\mathbf{y}) = 0$ in our case and $\partial\Omega$ is the boundary of Ω. Once we obtain the gradient of Ψ, we can easily compute the curl of Ψ as well (e.g., in 2D by 90° rotation), which yields the velocity.

Monte Carlo estimation To estimate the gradient of the stream function Ψ (or velocity $\mathbf{u}$), we can use the pointwise WoS gradient estimator (Section 2.3.3) for the aforementioned Poisson equation with Dirichlet conditions. To evaluate $\nabla_{\mathbf{x}} \Psi(\mathbf{x_0})$, the WoS algorithm estimates a recursive integral equation by recursively sampling a point $\mathbf{y}_k$ inside the largest sphere around the current point $\mathbf{x}_k$ and sampling another point $\mathbf{x}_{k+1}$ on the boundary of sphere to continue its recursion. A slight improvement we propose is that we use antithetic sampling to get the first boundary sample, meaning that in addition to $\mathbf{x}_1$ we always add an extra sample, $\mathbf{x}_1^m$, on the opposite side of the sphere (also applying WoS to it); see Fig. 5.3 for visual intuition. Omitting antithetic sampling can easily lead to a large variance when trying to compute the gradient of a constant non-zero function. Note that this WoS estimator is a pointwise estimator and it only requires the pointwise estimation of the vorticity as its input, similar to the case without any solid boundaries. Note that the choice to use WoS, instead of WoB, which we have discussed in the previous chapters, is solely because WoS was the only option we were aware of when developing this vorticity-based Monte Carlo fluid solver. WoB would serve as a drop-in replacement.

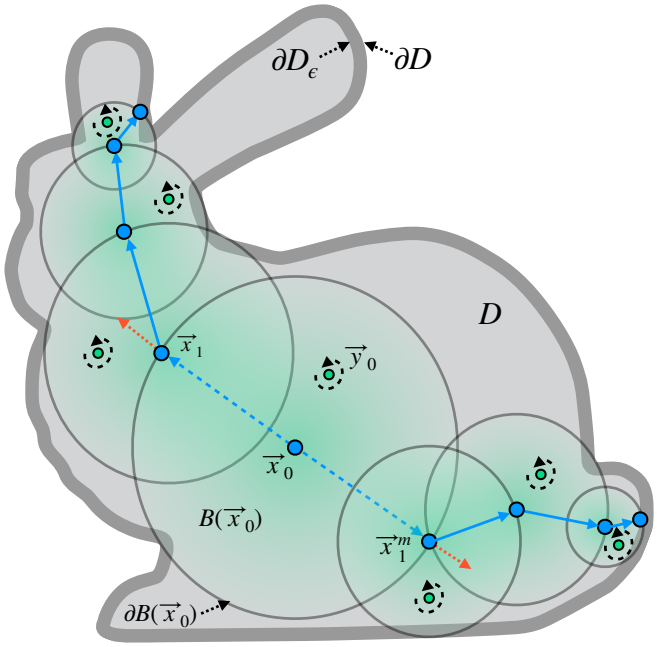


Figure 5.3: Walk on Spheres gradient computation. Consider the interior samples $\mathbf{y}_k \sim \partial B(\mathbf{x}_k)$ (green) and boundary samples $\mathbf{x}_{k+1} \sim B(\mathbf{x}_k)$ (blue). We sample two points $\mathbf{x}_1$ and $\mathbf{x}_1^m$ on the largest inscribed sphere centered at $\mathbf{x}_0$ using antithetic sampling and compute their respective normals (red arrows) at the boundary. From these sampled positions, we proceed with a standard WoS – tracing two opposing random paths – and end the recursion once we penetrate a threshold region ∂D_ϵ (dark grey) to compute $\hat{\Psi}(\mathbf{x}_1)$ and $\hat{\Psi}(\mathbf{x}_1^m)$.

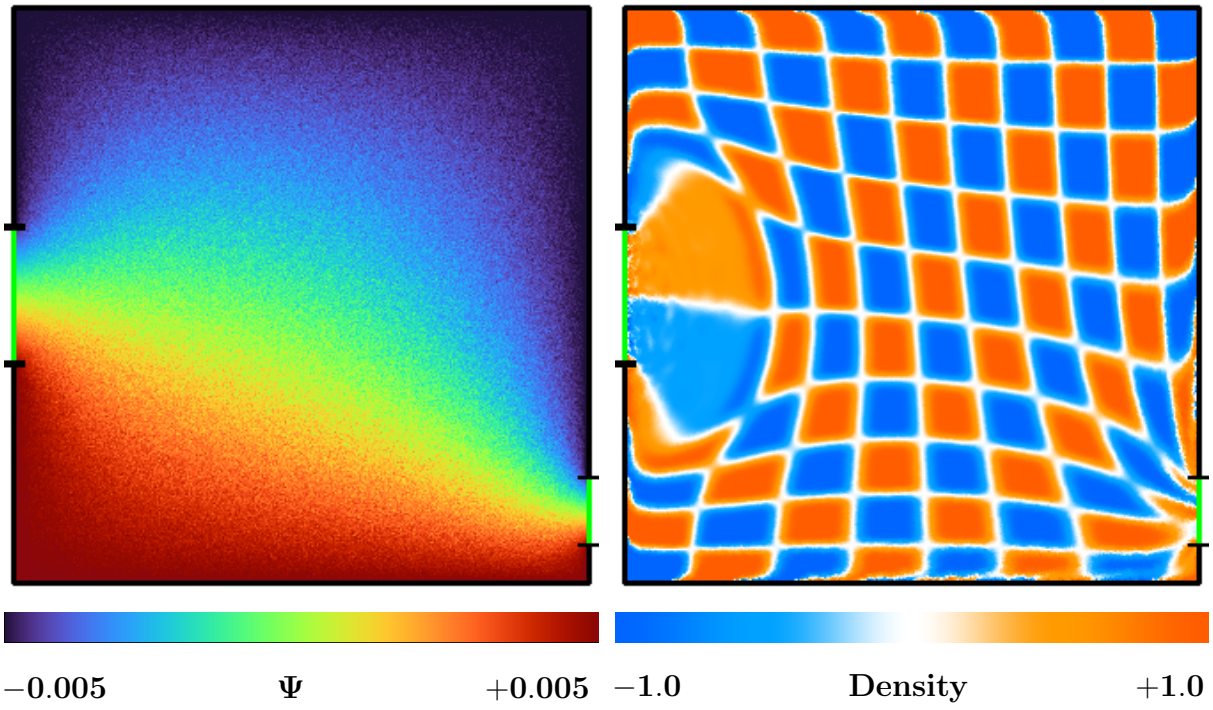


Figure 5.4: Inflow and outflow boundaries. Inflow and outflow regions are highlighted in green on the boundary of the domain. Left: the stream function corresponding to setting the top part of the solid wall to $\Psi = 0.005$ and the bottom to $\Psi = -0.005$. Ψ values at the inflow and outflow vary linearly, yielding constant normal velocity. Right: the motion of an advected checkerboard density field.

Other boundary conditions With a slight modification of the construction above, we can support inflow and outflow boundaries. By dividing the solid boundary into distinct pieces, each with a constant stream function value, and connecting them via linearly interpolated stream function values along the inflow/outflow segments of the boundary, we arrive at a stream function along the inflow/outflow whose gradient is parallel to the boundary (Fig. 5.4). Taking the (negative) curl recovers the velocity that yields the desired perpendicular (constant) inflow or outflow.

We can further generalize our inflow/outflow treatment to moving solid boundaries with prescribed velocities, again with free slip (Fig. 5.5). In 2D, the stream function along the boundary must satisfy the constraint

$$\mathbf{u}_b \cdot \mathbf{n} = \partial\Psi / \partial\hat{t}\,, \tag{5.8}$$

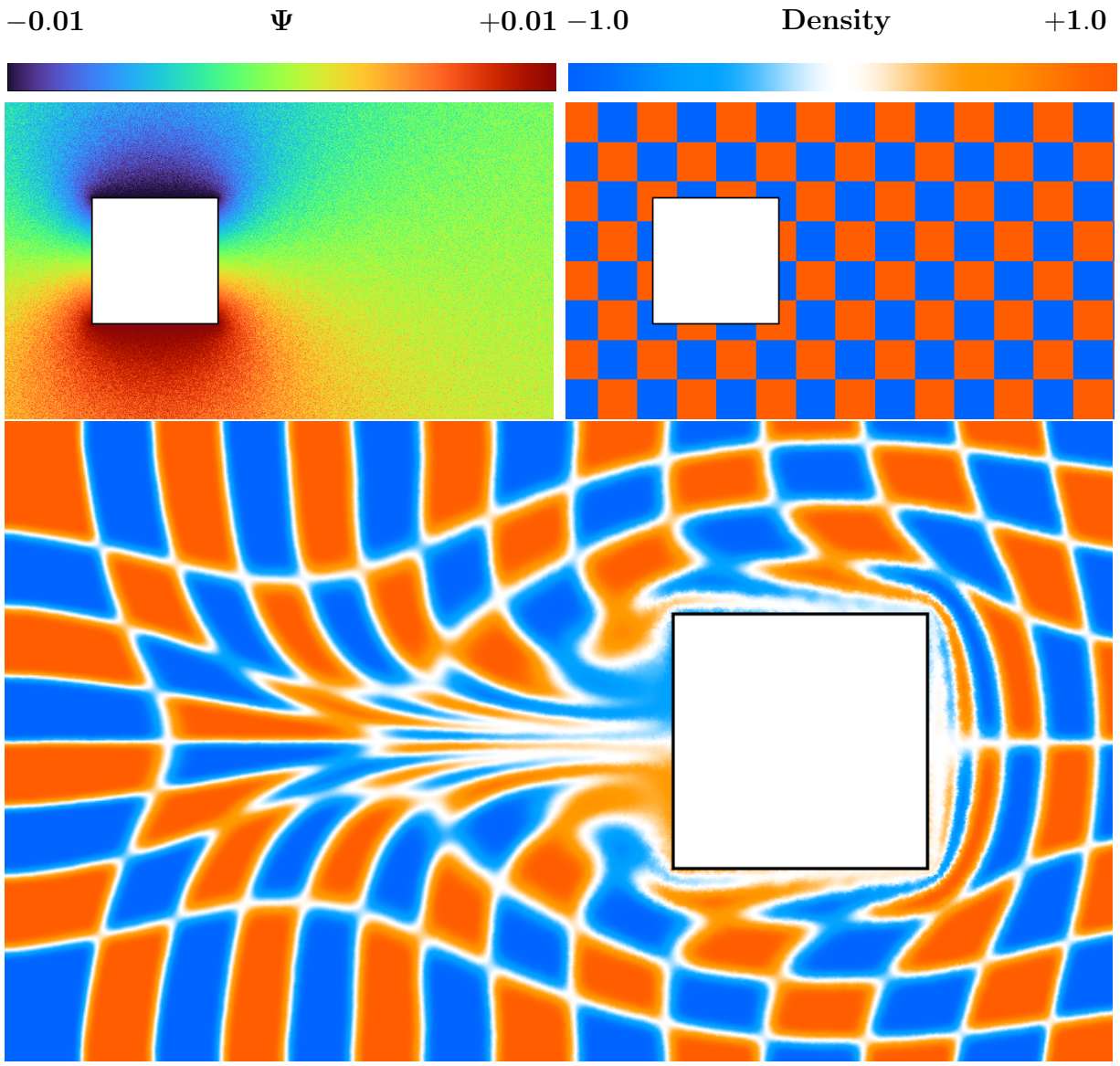


Figure 5.5: Moving obstacles. A square moves left to right at constant speed through the domain. The stream function (top left) and the initial density field (top right) to be advected, and the result after some time (bottom).

where $\mathbf{u}_b$ is the velocity on the boundary, and $\mathbf{n}$ and $\hat{t}$ are the unit normal and tangent vectors of the boundary.

We can choose any reference point on the solid surface and set it to some constant value Ψ_0, after which determining the stream function along the rest of the boundary (e.g., for a polygon with Ψ_i data at nodes $\mathbf{s}_i$) from the prescribed velocity field $\mathbf{u}_b$ requires integrating the expression above to obtain

$$\psi_{i+1} = \psi_i + \int_{\mathbf{s}_i}^{\mathbf{s}_{i+1}} (\mathbf{u}_b \cdot \mathbf{n}) \, \mathrm{d}\mathbf{s} \, . \tag{5.9}$$

Assuming a divergence-free solid boundary velocity field, the net flux along the surface will be zero; thus, the integration around its boundary loop back to $\mathbf{x}_0$ will yield the same starting value of Ψ.

For polygons and other simple shapes undergoing simple (e.g., rigid body) motions, we can obtain exact expressions of the stream functions on the obstacle boundaries [Bridson et al. 2007].

5.2.2 Vortex Stretching

In 3D, vorticity-based methods must evaluate the vortex stretching term, which arises from the deformation of the fluid. Note that we can skip this step for 2D problems because the vortex stretching term vanishes.

After operator splitting, the corresponding equation is $\frac{\partial \omega}{\partial t} = (\omega \cdot \nabla)\mathbf{u}$. Applying forward Euler discretization gives the pointwise update rule for the vorticity:

$$\omega \leftarrow \omega + \Delta t \, (\omega \cdot \nabla)\mathbf{u}. \tag{5.10}$$

The main challenge is estimating the second term in this update rule, $(\omega \cdot \nabla)\mathbf{u}$. One can approximate the this term in various ways: by finite differences, by applying Monte Carlo integration by evaluating the gradient of the velocity field through the Biot-Savart integral (Eq. 5.3) or through a Hessian WoS estimator [Sawhney and Crane 2020], or by converting the vorticity field to a vortex segment representation and advecting it according to the velocity field, as in the work of Zhang and Bridson [2014]. We adopt the last approach due to its superior stability in practice; we approximate the term $(\omega \cdot \nabla)\mathbf{u}$ by

$$[\omega(\mathbf{x}) \cdot \nabla] \, \mathbf{u}(\mathbf{x}) \approx \frac{|\omega(\mathbf{x})|}{h} \left[\mathbf{u}\left(\mathbf{x} + \overrightarrow{\Delta x}\right) - \mathbf{u}\left(\mathbf{x} - \overrightarrow{\Delta x}\right) \right], \tag{5.11}$$

where $\overrightarrow{\Delta x} = (h/2)\omega(\mathbf{x})$ and h is the user-defined length of the vortex segment. (Note that $\overrightarrow{\Delta x}$ is unrelated to the gradient of $\mathbf{x}$.) One can estimate the velocities at the ends of a vortex segment using an appropriate velocity estimator to get an estimate of $(\omega \cdot \nabla)\mathbf{u}$. See Section 5.5 for the implementation details.

5.2.3 Diffusion

When the viscosity coefficient ν is nonzero, we consider this diffusion step, which solves $\frac{\partial \omega}{\partial t} = \nu \nabla^2 \omega$. The exact solution to this problem is known in $\mathbb{R}^d$: the resulting vorticity field can be evaluated as the convolution of the input vorticity field with a Gaussian of zero mean and variance $2\nu\Delta t$ [Chorin 1973; Park and Kim 2005]. We use pointwise Monte Carlo estimation to approximate this convolution:

$$\omega(\mathbf{x}) \leftarrow \frac{1}{n_d} \sum_{i=1}^{n_d} \omega(\mathbf{x} + \sqrt{2\nu\Delta t}\, \xi_i), \tag{5.12}$$

where ξ_i are independent samples drawn from the d-dimensional standard normal distribution. While we use this scheme for the vorticity-based formulation, readers should be aware of alternative approaches that account for solid boundaries, as discussed in Section 5.3.4.

5.2.4 Advection

We now advect the vorticity field, after applying vortex stretching and diffusion, using the pointwise-estimated velocity field computed prior to these steps, as described in Section 5.2.1. We employ a *semi-Lagrangian* advection method, which estimates the new vorticity at a query point by tracing trajectories backward in time through the velocity field [Fedkiw et al. 2001; Stam 1999]. After discretizing in time with time step Δt, a basic semi-Lagrangian scheme with Forward Euler time integration of trajectories approximates the updated vorticity at position $\mathbf{x}$ by querying the input vorticity field as

$$\omega(\mathbf{x}) \leftarrow \omega(\mathbf{x} - \Delta t\, \mathbf{u}(\mathbf{x})). \tag{5.13}$$

This semi-Lagrangian update satisfies our pointwise evaluation requirement: we can estimate the vorticity after the step by evaluating the input velocities and vorticities at one point each. While the time-discretized equation above is identical to what appears in grid-based fluid solvers, we emphasize that the point where we evaluate the advected vorticity, $\mathbf{x}$, does not necessarily need to be aligned with discretized locations (e.g., grid

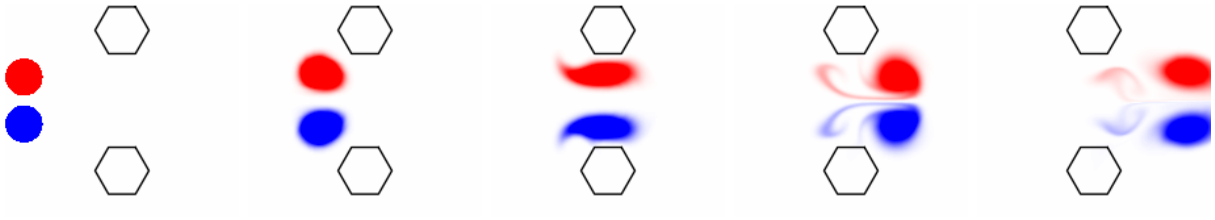
points), and when we evaluate the pre-advection velocity and vorticity at points $\mathbf{x}$ and $\mathbf{x} - \Delta t\, \mathbf{u}(\mathbf{x})$, we are not restricted to evaluation by interpolation of some grid data. Thus, the pointwise form offers flexibility that is unavailable with classical solvers. In practice, rather than the forward Euler method, we can use a higher-order time integration scheme (e.g., Runge-Kutta) to improve the accuracy of the traced trajectories. The adaptations of error-correcting schemes, such as MacCormack [Selle et al. 2008] and BFECC [Dupont and Liu 2007], are also straightforward; all of these schemes support pointwise evaluation.

5.3 Velocity-Based Monte Carlo Method

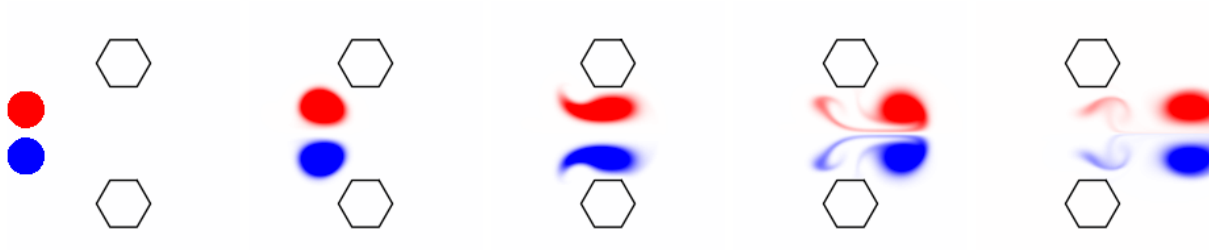
The other formulation we consider is the velocity-based formulation, which solves the Navier-Stokes equations using velocity and pressure variables. Since the introduction of velocity-based Eulerian grid fluid simulation techniques to graphics [Foster and Metaxas 1996; Stam 1999], researchers have improved on many aspects of the velocity-based formulation, adding features such as buoyancy modeling for smoke [Fedkiw et al. 2001; Foster and Metaxas 1997] and velocity divergence control for expansion/contraction and artistic effects [Feldman et al. 2003], as well as PIC/FLIP solvers [Brackbill and Ruppel 1986; Harlow 1962; Zhu and Bridson 2005] and advection-reflection solvers [Narain et al. 2019; Zehnder et al. 2018] for reducing numerical dissipation. These and many other improvements cannot be straightforwardly incorporated into the vorticity-based Monte Carlo fluid solver. Furthermore, Yin et al. [2023] recently showed that existing vorticity-based methods can fail to simulate harmonic velocity fields, leading to incorrect results when the fluid domain is not simply connected (e.g., is disjoint or has holes) (Fig. 5.6). Conveniently, velocity-based methods can capture these physics without any change.

Operator splitting with pointwise estimators Similar to the vorticity-based method, the basic formulation of our method follows the standard operator splitting framework [Stam 1999]. We solve Eq. 5.1 by taking discrete time steps with step size Δt. For each time step, we decompose the Navier-Stokes equations into four substeps:

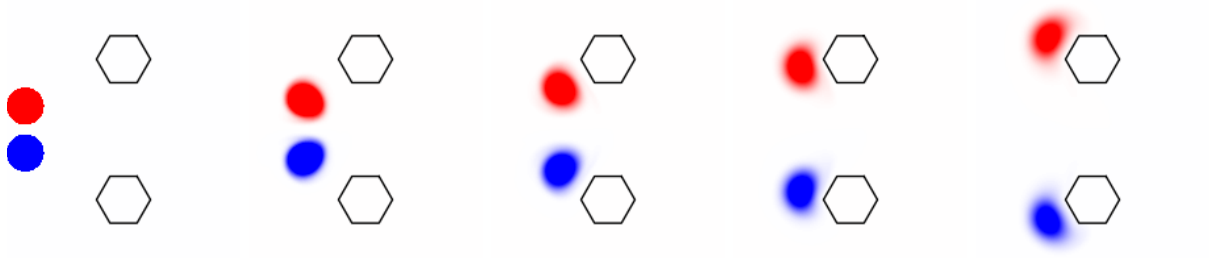
1. advection: $\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u}$,
2. external force integration: $\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f}$,
3. diffusion: $\frac{\partial \mathbf{u}}{\partial t} = \nu \nabla^2 \mathbf{u}$, and
4. projection: $\frac{\partial \mathbf{u}}{\partial t} = -\frac{1}{\rho}\nabla p$ such that $\nabla \cdot \mathbf{u} = 0$.



Correct Physics with Velocity-Based Monte Carlo Method



Correct Physics with Conventional Velocity-Based Method [Batty et al. 2007]



Incorrect Physics with Vorticity-Based Monte Carlo Method

Figure 5.6: Our velocity-based Monte Carlo fluid solver can naturally handle scenes for which the vorticity-based Monte Carlo method yields incorrect results. The velocity-based solver allows the red and blue smoke densities associated with a pair of vortices to flow between two obstacles (top), similar to the conventional non-Monte Carlo velocity-based method [Batty et al. 2007] (middle), while the vorticity-based method produces an incorrect result, in which smoke deviates around the outside of the obstacles (bottom).

Similarly to the vorticity-based method, we develop a pointwise estimator for each substep. This allows for the theoretically spatial-discretization-free solver (Section 5.4), and we can consider more practical variants that incorporate discrete structures to cache the intermediate velocity field (Section 5.5). Depending on how we introduce such discrete storage of the field, our formulation allows us to remove some of the excessive grid interpolation errors that are otherwise introduced in traditional solvers. Our formulation is agnostic to the underlying discretization of the velocity field, enabling the flexible integration of velocity-based techniques commonly used by traditional discretization-based solvers. These include the Stable Fluids-style grid-based methods [Stam 1999] and grid-particle hybrid PIC/FLIP methods.

Below, we formulate a pointwise estimate of the solution to each substep. To simplify notation, we use $\mathbf{u}_0$ to $\mathbf{u}_4$ to indicate the velocity field at each substep within a single time step. The initial velocity field for the current time step $\mathbf{u}_0$ is defined as the output velocity field $\mathbf{u}_4$ from the last time step. The advection step takes $\mathbf{u}_0$ as its input and outputs $\mathbf{u}_1$, and so on.

5.3.1 Advection

We employ semi-Lagrangian advection, which estimates the updated velocity at a point by tracing particle trajectories backward in time through the velocity field. For example, applying a forward Euler time discretization yields the update rule for the velocity at position $\mathbf{x}$:

$$\mathbf{u}_1(\mathbf{x}) \leftarrow \mathbf{u}_0(\mathbf{x} - \Delta t\, \mathbf{u}_0(\mathbf{x})). \tag{5.14}$$

This semi-Lagrangian update satisfies our requirement for pointwise evaluation, analogous to the advection of vorticity discussed in Section 5.2.4. The same considerations apply here, with the only difference being that we are now advecting the velocity field itself using the same (temporarily frozen) velocity field.

5.3.2 External Force Integration

In the presence of external forces leading to acceleration, such as a buoyancy force, we update the post-advection velocity using a forward Euler discretization:

$$\mathbf{u}_2(\mathbf{x}) \leftarrow \mathbf{u}_1(\mathbf{x}) + \Delta t\, \mathbf{f}(\mathbf{x}). \tag{5.15}$$

This expression is again a pointwise update of the velocity field, where $\mathbf{x}$ is not necessarily at a discretized location.

5.3.3 Projection

As the diffusion step is necessary only for viscous fluids and we can simply let $\mathbf{u}_3 \leftarrow \mathbf{u}_2$ for inviscid fluids, we first discuss the projection step. The objective of the projection step is to find the pressure field that projects out the divergent velocity mode of the input field. The post-projection velocity at $\mathbf{x}$ is given by

$$\mathbf{u}_4(\mathbf{x}) \leftarrow \mathbf{u}_3(\mathbf{x}) - \nabla \overline{p}(\mathbf{x}), \tag{5.16}$$

where $\overline{p}$ satisfies the Poisson equation

$$\nabla^2 \overline{p}(\mathbf{x}) = \nabla \cdot \mathbf{u}_3(\mathbf{x}), \tag{5.17}$$

and $\overline{p} = \frac{\Delta t}{\rho} p$. Since only $\nabla \overline{p}$ is required for the update in Eq. 5.16, the task is reduced to estimating the gradient of the solution of the Poisson equation; the value of $\overline{p}$ itself is not needed. In each subsection below, we discuss the integral formulation followed by its evaluation with Monte Carlo integration.

Without Solid Boundaries

To simplify the problem, we will first consider a case without any solid boundaries (i.e., the domain is unbounded with no obstacles inside it). Under this assumption, it is well-known that we can write the solution to the Poisson equation using the fundamental solution G of the Laplace operator. The fundamental solution G satisfies $\nabla^2 G(\mathbf{x}, \mathbf{y}) + \delta(\mathbf{x} - \mathbf{y}) = 0$ in the unbounded domain, where δ is the Dirac delta function. In 2D, $G(\mathbf{x}, \mathbf{y}) = -\frac{1}{2\pi} \log r$, and in 3D, $G(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi r}$, where $r = \|\mathbf{y} - \mathbf{x}\|_2$. We can then write the solution to Eq. 5.17 as a convolution of the source term of the Poisson equation and the fundamental solution:

$$\overline{p}(\mathbf{x}) = -\int_{\mathbb{R}^d} G(\mathbf{x}, \mathbf{y})\, \nabla_{\mathbf{y}} \cdot \mathbf{u}_3(\mathbf{y})\, \mathrm{d}\mathbf{y}, \tag{5.18}$$

where dimension $d = 2, 3$. We used the subscript $\mathbf{y}$ in $\nabla_{\mathbf{y}}$ to indicate that we perform differentiation with respect to $\mathbf{y}$, and we will use similar notation throughout this section. By applying the Laplacian to Eq. 5.18, one can confirm that $\overline{p}$ satisfies Eq. 5.17. To evaluate the gradient of $\overline{p}$, we take the gradient of Eq. 5.18:

$$\nabla_{\mathbf{x}} \overline{p}(\mathbf{x}) = -\int_{\mathbb{R}^d} \nabla_{\mathbf{x}} G(\mathbf{x}, \mathbf{y})\, \nabla_{\mathbf{y}} \cdot \mathbf{u}_3(\mathbf{y})\, \mathrm{d}\mathbf{y}. \tag{5.19}$$

This integral is still not suitable for numerical computation for our purposes, as we describe below. We therefore extend beyond the prior work by proposing a further transformation.

Since we typically consider storing information and performing volume integrals over a bounded simulation domain, we replace the (infinite) integral domain $\mathbb{R}^d$ in Eq. 5.19 with a bounded simulation domain Ω^s by assuming that the velocity divergence $\nabla \cdot \mathbf{u}_3$ is zero outside the simulation domain.

Attempting to use Eq. 5.19 in this form for Monte Carlo integration still requires explicit evaluation of velocity divergence inside the domain, whereas we desire a solver that takes only a pointwise-evaluated velocity field as input. There are a few possible approaches to obtain the required divergence. First, we could apply finite differences to the velocity field by accepting some errors. Second, we could differentiate the substep that precedes projection so that it outputs the necessary velocity divergence, in addition to the velocity itself. We propose instead a velocity-only design that we believe fits better in our Monte Carlo framework.

To eliminate the dependency on velocity divergence in Eq. 5.19, we use the identity $\nabla_{\mathbf{x}} G = -\nabla_{\mathbf{y}} G$ and apply integration by parts:

$$\nabla_{\mathbf{x}} \overline{p}(\mathbf{x}) = \int_{\Omega^s} \{\nabla_{\mathbf{y}} G(\mathbf{x}, \mathbf{y})\} \, \nabla_{\mathbf{y}} \cdot \mathbf{u}_3(\mathbf{y}) \, \mathrm{d}\mathbf{y} \tag{5.20}$$

$$\begin{aligned} &= -\int_{\Omega^s} \mathbf{H}(\mathbf{x}, \mathbf{y}) \, \mathbf{u}_3(\mathbf{y}) \, \mathrm{d}\mathbf{y} \\ &\quad - \int_{\partial\Omega^s} \{\nabla_{\mathbf{x}} G(\mathbf{x}, \mathbf{y})\} \, \mathbf{n}(\mathbf{y}) \cdot \mathbf{u}_3(\mathbf{y}) \, \mathrm{d}\mathbf{y}, \end{aligned} \tag{5.21}$$

where $\mathbf{n}$ is the outward unit normal. The Hessian of the fundamental solution denoted $\mathbf{H}$, is given by

$$\mathbf{H}(\mathbf{x}, \mathbf{y}) = \mathbf{S}(\mathbf{x}, \mathbf{y}) - \frac{\delta(\mathbf{r})}{d}\mathbf{I}, \qquad \mathbf{S}(\mathbf{x}, \mathbf{y}) = \frac{1}{|\partial B| r^{d+2}}(d\mathbf{r}\mathbf{r}^\intercal - r^2\mathbf{I}), \tag{5.22}$$

where $|\partial B|$ is the surface area of a unit $(d-1)$-sphere, $\mathbf{r} = \mathbf{y} - \mathbf{x}$, $r = \|\mathbf{r}\|_2$ and $\mathbf{I}$ is the identity matrix. While we use the notation above for ease of understanding, technically, it needs to be understood in the sense of the generalized (distributional) derivative, and interested readers can refer to the discussion by Frahm [1983] and Hnizdo [2011] for further

details. Substituting Eq. 5.22 into Eq. 5.21, we get

$$\begin{aligned}\nabla_{\mathbf{x}}\overline{p}(\mathbf{x}) &= -\int_{\Omega^s} \mathbf{S}(\mathbf{x},\mathbf{y})\,\mathbf{u}_3(\mathbf{y})\,\mathrm{d}\mathbf{y} + \frac{1}{d}\mathbf{u}_3(\mathbf{x}) \\ &\quad -\int_{\partial\Omega^s} \{\nabla_{\mathbf{x}} G(\mathbf{x},\mathbf{y})\}\,\mathbf{n}(\mathbf{y})\cdot\mathbf{u}_3(\mathbf{y})\,\mathrm{d}\mathbf{y} \\ &= -\int_{\Omega^s} \mathbf{S}(\mathbf{x},\mathbf{y})\,\{\mathbf{u}_3(\mathbf{y}) - \mathbf{u}_3(\mathbf{x})\}\,\mathrm{d}\mathbf{y} \\ &\quad -\int_{\partial\Omega^s} \{\nabla_{\mathbf{x}} G(\mathbf{x},\mathbf{y})\}\,\mathbf{n}(\mathbf{y})\cdot\{\mathbf{u}_3(\mathbf{y}) - \mathbf{u}_3(\mathbf{x})\}\,\mathrm{d}\mathbf{y}.\end{aligned} \tag{5.23}$$

Again, technically, the domain of the first integral should exclude an infinitesimal ball around $\mathbf{x}$ [Hnizdo 2011], but we use this simple notation for readability. In Eq. 5.23, to get the final expression, we replaced the original velocity field with a velocity field that is globally shifted by the constant velocity at point $\mathbf{x}$, $\mathbf{u}_3(\mathbf{x})$. The computed pressure gradient remains unchanged because the divergence of the shifted velocity field is the same as the original one. This global shift cancels the zeroth-order term of the Taylor expansion of $\mathbf{u}_3(\mathbf{y})$ around $\mathbf{x}$, and the remaining terms of the Taylor expansion multiplied by the function $\mathbf{S}$ will have a singularity of $O(1/r^{d-1})$ instead of the original $O(1/r^d)$ as $r \to 0$. This can be interpreted as a form of control variate. This lower-order singularity can be handled by an appropriate importance-sampling strategy, as we discuss next.

Monte Carlo estimation Now that we have an integral representation of the pressure gradient, we can define a Monte Carlo estimator for the pressure gradient. We sample N_V points $\mathbf{y}^i$ inside the simulation domain Ω^s and N_A points $\mathbf{y}^j$ on the simulation domain boundary $\partial\Omega^s$ using sampling strategies with probability density functions (PDFs) p_V and p_A, respectively, to get

$$\nabla_{\mathbf{x}}\overline{p}(\mathbf{x}) \approx \langle E_V(\mathbf{x})\rangle + \langle E_A(\mathbf{x})\rangle \tag{5.24}$$

where

$$\langle E_V(\mathbf{x})\rangle = -\frac{1}{N_V}\sum_{i=1}^{N_V} \frac{\mathbf{S}(\mathbf{x},\mathbf{y}^i)}{p_V(\mathbf{y}^i|\mathbf{x})}\left\{\mathbf{u}_3(\mathbf{y}^i) - \mathbf{u}_3(\mathbf{x})\right\} \tag{5.25}$$

$$\langle E_A(\mathbf{x})\rangle = -\frac{1}{N_A}\sum_{j=1}^{N_A} \frac{\nabla_{\mathbf{x}} G(\mathbf{x},\mathbf{y}^j)}{p_A(\mathbf{y}^j|\mathbf{x})}\,\mathbf{n}(\mathbf{y}^j)\cdot\left\{\mathbf{u}_3(\mathbf{y}^j) - \mathbf{u}_3(\mathbf{x})\right\}. \tag{5.26}$$

This formulation is an unbiased estimator for the pressure gradient if the probability that we sample any point with a nonzero contribution is nonzero. If the integrand is non-

negative everywhere, importance sampling according to the PDF that is roughly proportional to the integrand can decrease the variance of the estimator. While our integrands may contain negative values, we follow this idea to design our sampling strategy. In our implementation, for Eq. 5.25, to handle the singular integral, we draw samples such that PDF p_V is proportional to $1/r^{(d-1)}$ inside of the smallest ball around $\mathbf{x}$ that fully contains the entire simulation domain, and set zero contributions from the samples outside the simulation domain. This approach is equivalent to evaluating the integral we get by extending the original integral domain to the ball and putting zero integrands in the extended part. We also use antithetic sampling here: in addition to a point $\mathbf{y}^i$ sampled this way, we always sample the symmetric point with respect to $\mathbf{x}$, $2\mathbf{x} - \mathbf{y}^i$ to further reduce variance for smooth velocity fields. For Eq. 5.26, we uniformly sample points on the simulation boundary. Properly sampling the positive and negative contributions separately may further reduce the variance of the estimators [Chang et al. 2023; Owen 2013], but we leave it as future work. Equation 5.24 only requires the ability to evaluate the velocity at the position of sample points to perform projection, in contrast to traditional solvers that typically require a globally coupled linear system solve.

With Solid Boundaries

Next, we consider the case when solid object boundaries are involved (Fig. 5.7). We still solve Eq. 5.17 for $\nabla\overline{p}$, but with free slip boundary conditions,

$$\frac{\partial\overline{p}}{\partial\mathbf{n}} = \mathbf{n}\cdot(\mathbf{u}_3 - \mathbf{u}_s) \tag{5.27}$$

on solid boundaries, where $\mathbf{n}$ is the unit outward normal from the fluid domain Ω, $\frac{\partial}{\partial\mathbf{n}} = \mathbf{n}\cdot\nabla$ is a normal derivative, and $\mathbf{u}_s$ is the solid velocity. If the simulation domain is bounded, this problem is an interior Neumann problem for the Poisson equation, and otherwise, it is an exterior Neumann problem; we need a Monte Carlo solver that is capable of handling these problems.

The basic WoS [Sawhney and Crane 2020] can only handle Dirichlet problems, and its applicability to unbounded domain problems leaves some concerns about the termination of paths without additional bias. The vorticity-based Monte Carlo method (Section 5.2.1) arbitrarily terminates its Walk on Spheres paths after a few steps in its simulation, leaving some additional bias. Nabizadeh et al. [2021] suggested inverting the unbounded domain into a bounded one using the Kelvin transform so that paths always terminate, but using this approach would require very different treatment for bounded and unbounded domains.

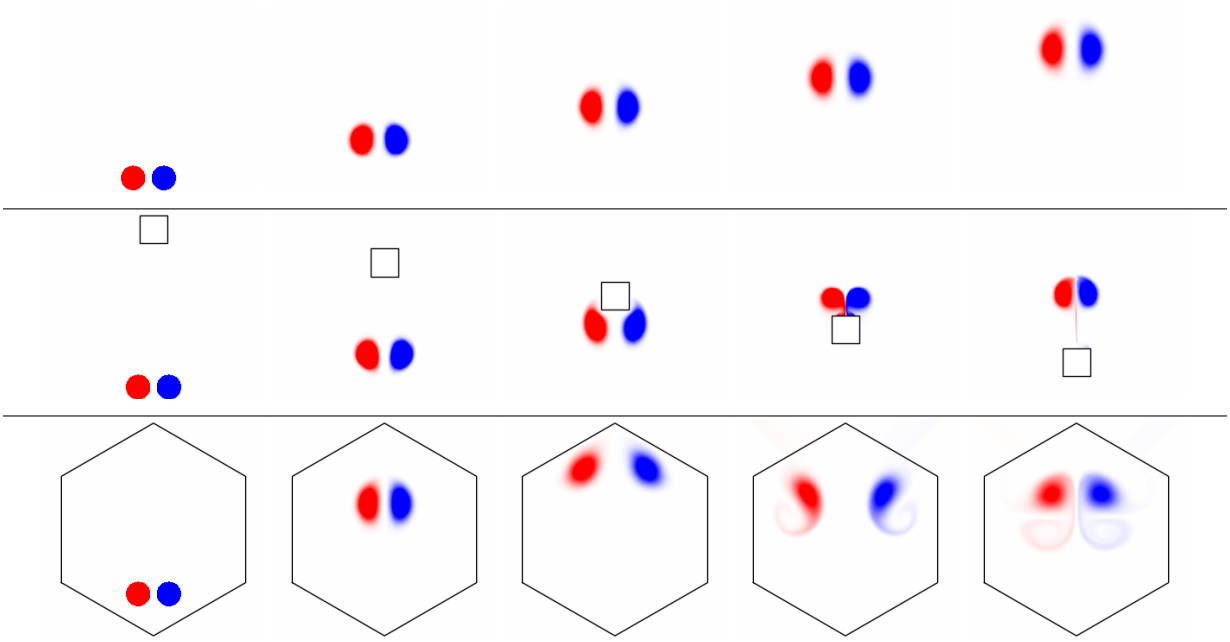


Figure 5.7: Our method can correctly simulate scenes with different boundary types (resolution 256^3). From the left, we simulate the colored smoke advected with the velocity field induced by vortex pairs moving with no obstacles (left), in an unbounded domain with a moving square obstacle (middle), and in a domain bounded by an obstacle (right). For each simulation, we show the time evolution from the left to the right, but the time stamps differ for each setup.

To solve Neumann problems, the Walk on Stars method [Sawhney and Miller et al. 2023] extended WoS with carefully-designed recursion relationships and tailored sampling techniques based on the extensions by Simonov [2008] and Ermakov and Sipin [2009]. For a Neumann problem in an unbounded domain, however, the Kelvin transform [Nabizadeh et al. 2021] is still required; it transforms a Neumann problem into a Robin problem, to which the application of Walk on Stars has not yet been attempted. Among others, WoB (Section 3.2) can be applied to Neumann boundary problems in both bounded and unbounded domains under a unified framework. The method builds upon a boundary integral equation (BIE) analogous to the rendering equation for light transport simulation [Kajiya 1986], and uses ray-tracing to solve the BIE. We use this method in our solver.

Following Sabelfeld and Simonov [1994], we can write the gradient of the solution to the Poisson equation of Eq. 5.17 with the boundary condition of Eq. 5.27 using the single layer potential formulation and define an integral equation for the unknown density function. For our application, these equations contain the volume integral terms arising from the source term in the Poisson equation of Eq. 5.17 specific to our problem. Similar to the case without solid boundaries, we transform these volume integral terms as follows. If the domain Ω is unbounded, we suggest replacing it with a bounded simulation domain Ω^s, assuming zero velocity divergence outside of the simulation domain; otherwise, we let $\Omega^s = \Omega$. With this definition, we have $\partial\Omega \subseteq \partial\Omega^s$ for the boundaries, and we distinguish these two sets of boundaries in the following equations. We also remove the explicit dependencies on the velocity divergence by integration by parts similarly to Eq. 5.21 to get

$$\begin{aligned}\nabla_{\mathbf{x}}\overline{p}(\mathbf{x}) = &\int_{\partial\Omega} \{\nabla_{\mathbf{x}} G(\mathbf{x},\mathbf{y})\} \left[\mu(\mathbf{y}) - \mathbf{n}(\mathbf{y})\cdot\{\mathbf{u}_3(\mathbf{y}) - \mathbf{u}_3(\mathbf{x})\}\right] \mathrm{d}\mathbf{y} \\ &- \int_{\Omega^s} \mathbf{S}(\mathbf{x},\mathbf{y}) \{\mathbf{u}_3(\mathbf{y}) - \mathbf{u}_3(\mathbf{x})\} \mathrm{d}\mathbf{y} \\ &- \int_{\partial\Omega^s\setminus\partial\Omega} \{\nabla_{\mathbf{x}} G(\mathbf{x},\mathbf{y})\} \mathbf{n}(\mathbf{y})\cdot\{\mathbf{u}_3(\mathbf{y}) - \mathbf{u}_3(\mathbf{x})\} \mathrm{d}\mathbf{y}\end{aligned} \tag{5.28}$$

for $\mathbf{x} \in \Omega$ and

$$\begin{aligned}\mu(\mathbf{x}) = &- \int_{\partial\Omega} 2\frac{\partial G}{\partial \mathbf{n}_{\mathbf{x}}}(\mathbf{x},\mathbf{y}) \left[\mu(\mathbf{y}) - \mathbf{n}(\mathbf{y})\cdot\{\mathbf{u}_3(\mathbf{y}) - \mathbf{u}_3(\mathbf{x})\}\right] \mathrm{d}\mathbf{y} \\ &+ \int_{\Omega^s} 2\mathbf{n}(\mathbf{x})^{\intercal}\mathbf{S}(\mathbf{x},\mathbf{y}) \{\mathbf{u}_3(\mathbf{y}) - \mathbf{u}_3(\mathbf{x})\} \mathrm{d}\mathbf{y} \\ &+ \int_{\partial\Omega^s\setminus\partial\Omega} 2\frac{\partial G}{\partial \mathbf{n}_{\mathbf{x}}}(\mathbf{x},\mathbf{y}) \mathbf{n}(\mathbf{y})\cdot\{\mathbf{u}_3(\mathbf{y}) - \mathbf{u}_3(\mathbf{x})\} \mathrm{d}\mathbf{y} \\ &+ 2\mathbf{n}(\mathbf{x})\cdot\{\mathbf{u}_3(\mathbf{x}) - \mathbf{u}_s(\mathbf{x})\}\end{aligned} \tag{5.29}$$

for $\mathbf{x} \in \partial\Omega$. Note Eq. 5.28 is a generalization of the strategy for the case without solid boundaries, as we can recover Eq. 5.23 by dropping the integrals over solid boundaries.

Monte Carlo estimation Based on Eq. 5.28 and Eq. 5.29, we design a (biased) WoB Monte Carlo estimator. Note that Eq. 5.29 contains the unknown density function μ itself on the left-hand side of the equation and in one of the integrands on the right-hand side, making it a recursive integral equation similar to the rendering equation [Kajiya 1986] used in path tracing for light transport simulation rather than a simple non-recursive one. Following Section 3.2.1, we truncate the recursion after M steps and multiply the contribution of the longest path by 0.5 to design a biased estimator.

We can consider various sampling strategies to estimate the solution to the truncated recursive integral equation with Monte Carlo methods. We use a forward estimator with boundary value caching, akin to the virtual point light (VPL) method [Keller 1997] in rendering, to solve this in our implementation: to sample a path consisting of multiple boundary points, we randomly sample a boundary point first, take a random walk on randomly sampled boundary points from there, and finally connect them to each individual evaluation point. The following describes the details of our sampling scheme. Readers not interested in these technical details may proceed directly to Section 5.3.4.

Let $p_U(\mathbf{x}_0)$ denote the PDF for uniform sampling on the solid boundary $\partial\Omega$, and we generate a point $\mathbf{x}_0$ with it. Next, we use a PDF $p_R(\mathbf{x}_{m+1}|\mathbf{x}_m)$ to generate samples according to the following sampling strategy: Starting from a point $\mathbf{x}_m$, we uniformly sample a direction from the unit hemisphere centered at $\mathbf{x}_m$, and then uniformly select one of the intersection points between the resulting line and the solid boundary $\partial\Omega$ to obtain the next point $\mathbf{x}_{m+1}$. By recursively applying this sampling strategy, we generate a sequence of points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{M+1}$ to form a path, conditioned on the previously sampled points. We use this strategy because the PDF associated with this specific forward sampling strategy is known to be proportional to the integral kernel $\frac{\partial G}{\partial \mathbf{n}_\mathbf{x}}$ we have in Eq. 5.29.

Compared to the simplest formulation for the Laplace equation, we have some additional non-recursive terms in Eq. 5.28 and Eq. 5.29, and we need to consider how to sample such terms, too. To do so, we first define two estimators for length m contributions, $\langle \mu_m^1(\mathbf{x}) \rangle$ and $\langle \mu_m^2(\mathbf{x}) \rangle$, recursively, based on Eq. 5.29: we define the base case for length 1 contributions as

$$\langle \mu_1^1(\mathbf{x}_1) \rangle = \frac{2\frac{\partial G}{\partial \mathbf{n}_\mathbf{x}}(\mathbf{x}_1, \mathbf{x}_0)}{p_R(\mathbf{x}_1|\mathbf{x}_0)} \mathbf{n}(\mathbf{x}_0) \cdot \{\mathbf{u}_3(\mathbf{x}_0) - \mathbf{u}_3(\mathbf{x}_1)\}, \tag{5.30}$$

$$\langle \mu_1^2(\mathbf{x}_0) \rangle = 2\mathbf{n}(\mathbf{x}_0) \cdot \{-\langle E_V(\mathbf{x}_0) \rangle - \langle E_A(\mathbf{x}_0) \rangle + \mathbf{u}_3(\mathbf{x}_0) - \mathbf{u}_s(\mathbf{x}_0)\}, \tag{5.31}$$

where $\langle \mu_1^1(\mathbf{x}_1) \rangle$ corresponds to the non-recursive contribution from the first integral in Eq. 5.29 and $\langle \mu_2^1(\mathbf{x}_0) \rangle$corresponds to the rest of the non-recursive contributions in Eq. 5.29. Then, we define the contributions from longer paths as

$$\langle \mu_{m+1}^1(\mathbf{x}_{m+1}) \rangle = \frac{-2\frac{\partial G}{\partial \mathbf{n}_\mathbf{x}}(\mathbf{x}_{m+1}, \mathbf{x}_m)}{p_R(\mathbf{x}_{m+1}|\mathbf{x}_m)} \langle \mu_m^1(\mathbf{x}_m) \rangle, \tag{5.32}$$

$$\langle \mu_{m+1}^2(\mathbf{x}_m) \rangle = \frac{-2\frac{\partial G}{\partial \mathbf{n}_\mathbf{x}}(\mathbf{x}_m, \mathbf{x}_{m-1})}{p_R(\mathbf{x}_m|\mathbf{x}_{m-1})} \langle \mu_m^2(\mathbf{x}_{m-1}) \rangle. \tag{5.33}$$

Note in particular, when we use the above-mentioned uniform line intersection sampling for $p_R(\mathbf{y}|\mathbf{x})$,

$$\frac{2\frac{\partial G}{\partial \mathbf{n}_\mathbf{x}}(\mathbf{x}, \mathbf{y})}{p_R(\mathbf{x}|\mathbf{y})} = \kappa(\mathbf{y}, \mathbf{x}) \cdot \mathrm{sgn}(\mathbf{n}(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{y})), \tag{5.34}$$

where $\kappa(\mathbf{y}, \mathbf{x})$ is the number of intersection points that the line that goes through the two points has, excluding point $\mathbf{y}$, and sgn is the sign function. Then, based on Eq. 5.28, we can estimate the pressure gradient by taking N_P sample paths in addition to the terms in Eq. 5.24:

$$\begin{aligned}
\langle \nabla_\mathbf{x} \overline{p}(\mathbf{x}) \rangle = &\langle E_V(\mathbf{x}) \rangle + \langle E_A(\mathbf{x}) \rangle \\
&+ \frac{1}{N_P} \sum_{k=1}^{N_P} \Bigg[-\frac{\nabla_\mathbf{x} G(\mathbf{x}, \mathbf{x}_0^k)}{p_U(\mathbf{x}_0^k)} \mathbf{n}(\mathbf{x}_0^k) \cdot \left\{ \mathbf{u}_3(\mathbf{x}_0^k) - \mathbf{u}_3(\mathbf{x}) \right\} \\
&+ \frac{\nabla_\mathbf{x} G(\mathbf{x}, \mathbf{x}_{M+1}^k)}{2 p_U(\mathbf{x}_0^k)} \langle \mu_M^1(\mathbf{x}_{M+1}^k) \rangle + \frac{\nabla_\mathbf{x} G(\mathbf{x}, \mathbf{x}_M^k)}{2 p_U(\mathbf{x}_0^k)} \langle \mu_M^2(\mathbf{x}_M^k) \rangle \\
&+ \sum_{m=1}^{M-1} \frac{\nabla_\mathbf{x} G(\mathbf{x}, \mathbf{x}_{m+1}^k)}{p_U(\mathbf{x}_0^k)} \langle \mu_m^1(\mathbf{x}_{m+1}^k \rangle + \frac{\nabla_\mathbf{x} G(\mathbf{x}, \mathbf{x}_m^k)}{p_U(\mathbf{x}_0^k)} \langle \mu_m^2(\mathbf{x}_m^k) \rangle \Bigg].
\end{aligned} \tag{5.35}$$

On the right-hand side, the first line estimates the last two integrals in Eq. 5.28, the second line estimates the non-recursive terms in the first integral, the third line estimates the longest path contributions, and the last line estimates the shorter path contributions. Note that while we do not explicitly indicate so, the contributions from paths of any length $\langle \mu_m^1(\mathbf{x}_{m+1}^k) \rangle$ and $\langle \mu_m^2(\mathbf{x}_m^k) \rangle$ implicitly depend on the sampled boundary point $\mathbf{x}_0^k$ due to the sampling strategy we employ.

To use the estimator of Eq. 5.35, a straightforward approach would be to generate sample paths for each individual evaluation point, which has a high computational cost. Instead, we use a boundary value caching strategy similar to the virtual point light method in rendering [Keller 1997]. We first compute the contributions of N_P subpaths up to the

point before we connect them to the evaluation points and cache them at $M+1$ boundary points per path. We then connect all of them to all evaluation points. This significantly increases the effective number of sample paths per evaluation point without significantly increasing the computational cost, while introducing a correlation between estimates at different evaluation points. Even this correlation can be preferable for our application because it guarantees that the contributions from these paths change smoothly across evaluation points.

In Fig. 3.5, we applied a similar caching technique to their Neumann problem WoB solver based on the same single-layer boundary integral formulation as well, but we used a backward estimator, which works only with a less efficient resampled importance sampling strategy. In contrast, the method here can utilize a forward estimator with more efficient line intersection importance sampling. This is at the cost of increasing storage per cache point by a small constant multiplication factor and increasing computation per evaluation point when we sum up the contributions from all cache points.

Similar to the problem without solid boundaries, for the estimation of volume integrals $\langle E_V(\mathbf{x})\rangle$ in Eq. 5.31 and Eq. 5.35, we use the importance sampling strategy so PDF p_V is proportional to $1/r^{(d-1)}$, and let $\mathbf{S}(\mathbf{x},\mathbf{y}) = \mathbf{0}$ for all $\mathbf{y}$ outside the simulation domain. Additionally, we need to set $\mathbf{S}(\mathbf{x},\mathbf{y}) = \mathbf{0}$ for all $\mathbf{y}$ inside of solid obstacles, too. We perform this insidedness test by casting a ray from point $\mathbf{y}$ in a random direction and taking a sum of its intersection signs. This strategy can be considered a Monte Carlo estimator for the generalized winding number [Jacobson et al. 2013]. We use uniform sampling for $\langle E_A(\mathbf{x})\rangle$.

For all 2D examples in Section 5.7.2, except Fig. 5.19, we use $N_V = N_A = 5 \cdot 10^5$ for direct contributions to each velocity evaluation point. For indirect (path) contributions, we use $N_P = 5 \cdot 10^5$ paths with path length $M = 4$. For each path, we use a reduced sample count of $N_V = N_A = 10$ within Eq. 5.31.

5.3.4 Diffusion

Simulating viscous fluids requires an additional diffusion step before the projection step. The set of equations we solve here is the constant-coefficient diffusion equation,

$$\begin{aligned} \frac{\partial \overline{\mathbf{u}}(\mathbf{x},s)}{\partial s} &= \nu \nabla^2 \overline{\mathbf{u}}(\mathbf{x},s) && \text{for } \mathbf{x} \in \Omega, s > 0, \\ \overline{\mathbf{u}}(\mathbf{x},s) &= \mathbf{u}_s(\mathbf{x}) && \text{for } \mathbf{x} \in \partial\Omega, s > 0, \quad \text{and} \\ \overline{\mathbf{u}}(\mathbf{x},0) &= \mathbf{u}_2(\mathbf{x}) && \text{for } \mathbf{x} \in \Omega, \end{aligned} \tag{5.36}$$

and the output of this diffusion step is $\mathbf{u}_3(\mathbf{x}) = \overline{\mathbf{u}}(\mathbf{x}, \Delta t)$. The time s here represents the time within each diffusion time step. We use the overbar to indicate that the velocity variable $\overline{\mathbf{u}}$ takes time s as defined here, and differs from the other sections. The solid velocity gives the no-slip boundary condition, and the output from the preceding external force integration step gives the initial conditions.

When there are no solid boundaries or the time step size is small enough to ignore boundary effects, we can omit the second equation describing the boundary condition. In such a case, evaluating a convolution of the input velocity field with a Gaussian function would give the exact solution to the diffusion equation. For the diffusion of the vorticity field rather than the velocity field, we used such a convolution-based approach to model diffusion in Section 5.2.3. However, this approach breaks down when the time step size is large enough that we require the explicit treatment of solid boundaries. Traditional grid-based solvers, similarly, cannot rely on a simple Gaussian filter in the presence of obstacles with a large time step size and must instead solve a globally coupled linear system for viscosity [Bridson 2015].

To address this problem, we utilize WoB for the diffusion equation (Section 4.1) to solve Eq. 5.36 component-wise. Similarly to WoB for projection, the diffusion WoB is a pointwise estimator and is a natural generalization of the case when there are no boundaries. One difference is that the BIE and the WoB steps are now defined in the space-time domain, and we must sample paths from the point where we want to evaluate the solution with time Δt towards the initial time within the time step, in the negative time direction.

Notably, for the diffusion WoB, the recursion depth does not need to be predefined. By sampling the space-time paths using a PDF proportional to the integral kernel of the integral equation for the diffusion equation, we can terminate recursions when the sampled time is negative. In our experiments, we observed that the diffusion step is cheaper than the projection step, and we did not attempt any algorithmic improvements, such as the reuse of subpaths. However, there remain many choices for sampling, and various efficiency improvement strategies should also apply here.

5.4 Spatial Discretization-Free Solver

We have so far discussed pointwise estimators for the substeps in both the vorticity- and velocity-based formulations. We now describe how to combine these estimators to compute the vorticity (and velocity, respectively) of a dynamic fluid at a prescribed location $\bar{\mathbf{x}}$ and time $\bar{t}$. The key idea is to recursively trace backward in time from the space-time query

point to the initial time $t = 0$, where the fluid's initial conditions are defined, using repeated applications of the pointwise estimators.

Consider a backtracing scenario for the vorticity-based formulation without viscosity in $\mathbb{R}^2$. To compute the vorticity at a query point $(\bar{\mathbf{x}}, \bar{t})$ via vorticity advection (Section 5.2.4), we must first estimate the velocity at this location. This velocity estimate, in turn, requires querying the vorticity field at several locations from the even more previous time step (Section 5.2.1). The vorticity values at those points again depend recursively on even earlier states of the field. Once this recursion bottoms out at $t = 0$, where the vorticity field is known, we have the estimated velocity at the original query point. We then update the vorticity value at the original query point by advecting this point to its backward position using the estimated velocity and evaluating the vorticity there, proceeding recursively until we reach the initial condition of the system (at $t = 0$). The final estimate of the vorticity value equals this estimate. Throughout this process, we never store the vorticity or velocity fields globally, nor do we introduce any spatial discretization.

This recursive strategy naturally extends to incorporate viscosity and vortex stretching in the vorticity-based formulation, as well as to the velocity-based formulation.

Discussion

In this algorithm, we are tracing flow characteristics backward in time for potentially long periods, and this idea is reminiscent of a family of recent Eulerian methods [Hachisuka 2005; Qu et al. 2019; Sato et al. 2018] that utilize characteristic mapping [Mercier et al. 2013; Tessendorf and Pelfrey 2011]. In contrast to these approaches, the proposed method is completely free from spatial discretization.

To our knowledge, Monte Carlo backtracing is the first numerical method to provide a *pointwise* solution of the fluid equations in both space and time. Given the benefits of Monte Carlo methods – namely, the ability to trade spatial discretization errors typical of alternative solvers for variance in the Monte Carlo estimate and the capability to apply various variance reduction techniques, Monte Carlo backtracing may be an attractive alternative. Moreover, when combined with physically-based Monte Carlo rendering methods (e.g., path tracing), the variance in the fluid solver *and* the renderer may likely lead to more perceptually pleasant errors [Cook 1986].

One important caveat in the formulation described thus far is that the recursive evaluations of the vorticity at each backtraced sample lead to an *exponential* computational complexity. Even with some advanced path reuse strategies, we would expect the cost to be at least linear. Unlike most rendering problems, where linearity will hold, due to the

nonlinearity of the advection term, we cannot simply take one sample to estimate the vorticity in the nested estimator. This major problem is implicitly woven into our formulation, but we will devise solutions that alleviate it in Section 5.5.

5.5 Practical Implementation with Caching

As outlined earlier, a naive implementation of our Monte Carlo Fluid algorithms suffers from an exponential increase in vorticity or velocity samples—all of which must be recursively evaluated—with respect to time. We offer a strategy to circumvent this problem using a discrete cache structure. Similar to dynamic programming solutions to recursive problems, we store and reuse the vorticity or velocity field as it is computed. We can proceed in two ways.

The first is to fill a spatio-temporal grid using the same recursion as before with respect to a position and time of interest. Doing so will gradually fill out a spatiotemporal "cone" of earlier data, rather than evaluating the whole domain for all time steps. This approach may be ideal when one wishes to evaluate the simulation backwards in time, e.g., in a view-dependent simulation.

Alternatively, the second option is to use a single uniform spatial grid that stores the field value computed at the center of each cell at the preceding time step. In essence, at every time step, we query this cache to retrieve the values from the previous time step. This strategy thus makes our method more similar to traditional discretization-based methods. Still, there is one fundamental difference; while we could always choose to cache the field after each substep within a time step, rather than after each substep, the pointwise estimation capability of our approach allows us to avoid some of the excessive field caching between substeps and reduce the associated errors. Also, since we do not evaluate any finite differences on the stored data, we do not need to use a staggered grid [Harlow and Welch 1965]. We will use this second approach in all of our results. In particular, for the vorticity-based variant, we always use caching on a per-time-step basis. For the velocity-based variant, we default to caching after each substep but also test other options for caching timings. Moreover, for problems involving solid boundaries, the properties of the underlying Monte Carlo PDE solvers would still carry over to our solver: it can handle complex obstacles without cut-sell or conforming meshing, is flexible with respect to geometric boundary representations, is robust to noise in the input geometry, and is easily parallelizable.

We will also show in Section 5.6, for the vorticity-based variant, that this caching method conveniently offers a way to generate importance-sampled Monte Carlo samples

distributed according to the vorticity field, reducing variance and increasing sample efficiency.

For the vorticity-based scheme, to evaluate the stretching term (Eq. 5.11) for 3D simulations, we choose h to be the grid resolution in the absence of boundaries, and in the presence of boundaries, we use the minimum of the grid resolution and twice the closest distance to any boundary. For this stretching term computation, we retrieve the cached velocities in practice, rather than using the pointwise velocity estimator, because bilinear or trilinear interpolation smoothes out the velocity field, which reduces noise when computing the stretching term, and also saves computational time.

5.6 Variance Reduction

Our Monte Carlo approaches open the opportunity to exploit a plethora of *variance reduction methods* within the context of fluid simulation. Indeed, we have already presented the antithetic sampling method for the WoS estimator (Section 5.2.1) and antithetic sampling, control variate, and importance sampling for the projection step (Section 5.3.3) to reduce variance. We also present a few additional basic yet practical variance reduction methods for the vorticity-based formulation, as examples of what is possible with our Monte Carlo approach. Section 5.8 discusses further potential opportunities.

Importance sampling One of the most popular and basic variance reduction methods is importance sampling. The idea is to generate samples according to a given probability density function (PDF) to reduce the variance. A well-chosen PDF can drastically reduce the variance, and thus the error of the estimators, and we can come up with a couple of such PDFs that are well-suited to our setting.

The first option is to generate samples proportionally to the integral kernel; the Biot–Savart kernel in the Biot–Savart case (without prescribed boundaries) and the Green's function in the WoS case (with prescribed boundaries). Sampling according to either of those functions is trivial. This importance sampling works especially well for WoS, but not always for the Biot–Savart case.

For the Biot–Savart case, we observed that this PDF can be only marginally better than uniform sampling in case non-zero vorticities are concentrated in a small region. The second option in this case is to directly generate samples according to the magnitude of the vorticity field. Since we are already caching vorticities to overcome the exponential cost of recursion, we can utilize this cache to perform importance sampling. Fig. 5.14 illustrates

an equal sample comparison between these two options. In this example, the baseline, which uses sampling according to vortices, produced a more accurate result (e.g., long and thin features of vorticities) than the alternative of sampling according to the Biot–Savart kernel.

Control variates A sequential nature of fluid simulation fits well with the method of control variates. The method of (difference) control variates utilizes another analytically integrable function (a *control variate*) to estimate only the difference between this function and the original integrand via Monte Carlo integration. This method can reduce variance when the original integrand and the control variate are correlated. In our application, to estimate the velocity using the Biot–Savart law (Eq. 5.3), we can use *the vorticity field from the previous time step* as a control variate, as follows:

$$\begin{aligned}\mathbf{u}(\mathbf{x},t) &= \int_\Omega [\omega(\mathbf{y},t)-\omega(\mathbf{y},t-\Delta t)]\times K(\mathbf{x}-\mathbf{y})\,\mathrm{d}\mathbf{y} + \int_\Omega \omega(\mathbf{y},t-\Delta t)\times K(\mathbf{x}-\mathbf{y})\,\mathrm{d}\mathbf{y} \\ &= \int_\Omega [\omega(\mathbf{y},t)-\omega(\mathbf{y},t-\Delta t)]\times K(\mathbf{x}-\mathbf{y})\,\mathrm{d}\mathbf{y} + \mathbf{u}(\mathbf{x},t-\Delta t).\end{aligned} \tag{5.37}$$

Note that having already estimated $\mathbf{u}(\mathbf{x},t-\Delta t)$ from $\omega(\mathbf{y},t-\Delta t)$ in the previous time step, this term is available in the current time step. Unlike a typical application of the method of control variates, however, this term $\mathbf{u}(\mathbf{x},t-\Delta t)$ is only an estimation with some variance, not an analytical integration of $\omega(\mathbf{y},t-\Delta t)$. When $\mathbf{u}(\mathbf{x},t-\Delta t)$ is similarly estimated by using $\omega(\mathbf{x},t-2\Delta t)$ as the control variate, this approach may not reduce variance overall. We circumvent this issue by estimating the initial velocity $\mathbf{u}(\mathbf{x},0)$ using a higher sample count than the rest. This approach will propagate variance reduction via this control variate at $t=0$ all the way to the current time, without increasing the sample count in any other time than $t=0$.

We generate samples according to the magnitude of the difference of vorticity fields at two consecutive time steps to evaluate the first integral in Eq. 5.37, and add the cached velocity from the previous time step. Due to the high correlation between the vorticity fields over time, we can expect that the variance of the estimator for $\omega(\mathbf{y},t)-\omega(\mathbf{y},t-\Delta t)$ is smaller than that of $\omega(\mathbf{y},t)$. Thus, this method greatly reduces the variance of our velocity estimate, given that the variance of the initial velocity field is low enough. Fig. 5.13 illustrates an equal sample comparison when we have the control variate enabled and disabled, and the method of control variates works well as expected. We applied this method only to the 3D scenes (without boundaries), where we expect significant reductions in sampling cost.

Note the control variate we used in Eq. 5.37 is not the general formula for control variates. The general form employs a combination of the estimator and control function with control parameters. It would be interesting to investigate how an optimal control parameter could be introduced in our setting to further reduce the estimator variance.

5.7 Results

5.7.1 Vorticity-Based Method

Comparison to Existing Methods

We compare our method to a pair of representative existing methods (Fig. 5.8): a particle-based solver that implements the vortex particle method of Park and Kim [2005] – using the Biot–Savart law to forward integrate vortex particle trajectories (2500 particles) while treating free slip boundaries with a panel method (80 panels) – and a (hybrid) grid-based FLIP gas solver [Bridson 2015; Zhu and Bridson 2005] with grid resolution $n_{\mathrm{x}} = 50$, 6 FLIP particles per grid cell, and $\Delta t = 0.05$. We visualize the vorticity field, in blue (positive) and orange (negative). We initialize the methods with the same vector field. Our simulation output is in strong agreement with the two baselines, suggesting that it indeed generates a valid solution to the fluid equations.

Our method using WoS runs at an average of 21 seconds per frame (with a multi-threaded CPU implementation). Both the vortex particle method and FLIP run orders of magnitude faster (i.e., roughly 0.005 and 0.25 seconds, respectively). Sawhney and Crane [2020] also noted a similar performance gap in their comparisons between WoS and finite-element methods for geometry processing. Despite this significant room for improvement in computation time, our method does have some fundamental advantages. First, the lack of reliance on a linear solve and the pointwise nature of our method make the computation easily parallelizable and GPU-friendly (as evidenced by our ShaderToy and CUDA implementations, discussed below). Second, traditional particle methods rely on nearest neighbor search to interpolate values, which can become quite expensive with many particles; this limitation does not exist in our method. Third, grid-based solvers generally suffer from numerical diffusion, as evidenced in the fluctuations of colors in Fig. 5.8, which is less evident in our method due to its pointwise nature. Last, our formulation based on WoS enables handling of noisy boundaries with no additional effort, as we discuss later. Similar to its application in geometry processing [Sawhney and Crane 2020], a further study

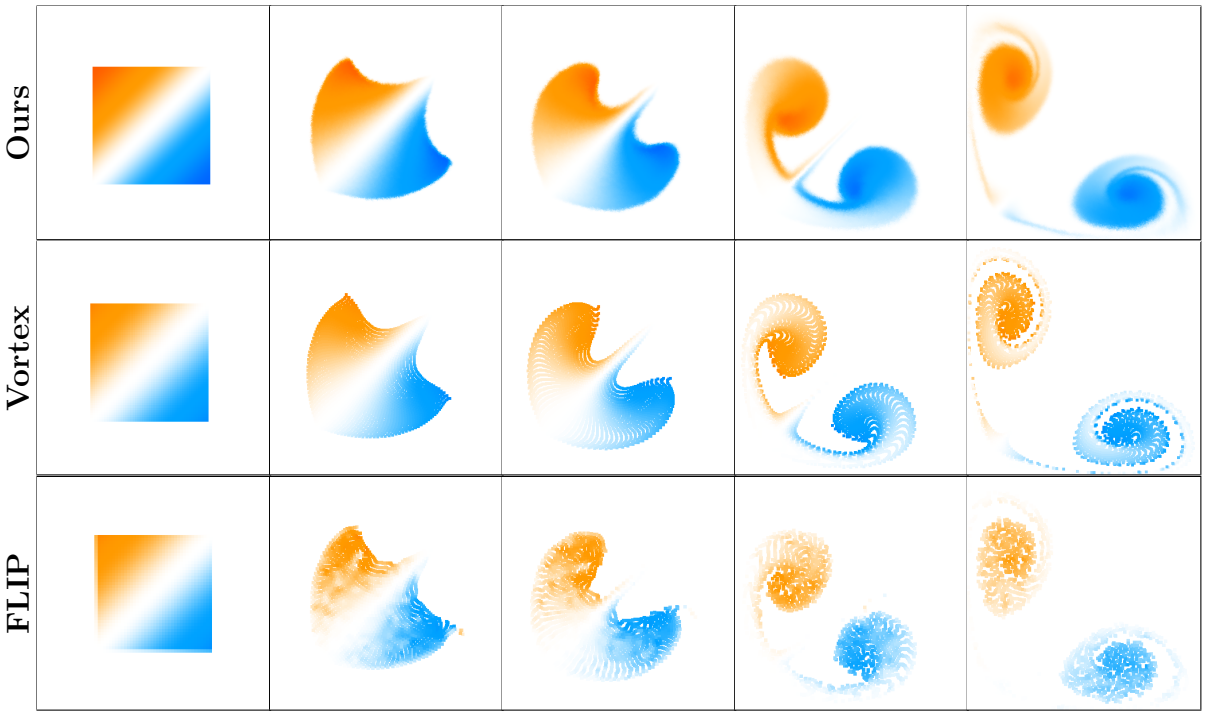


Figure 5.8: Comparison with standard solvers. Left to right, $t = 0, 1, 2, 5, 10$ seconds. Our method (top) produces results that are consistent with the vortex particle (middle) and hybrid FLIP (bottom) baselines. Colors visualize the signed vorticity at each (particle) position.

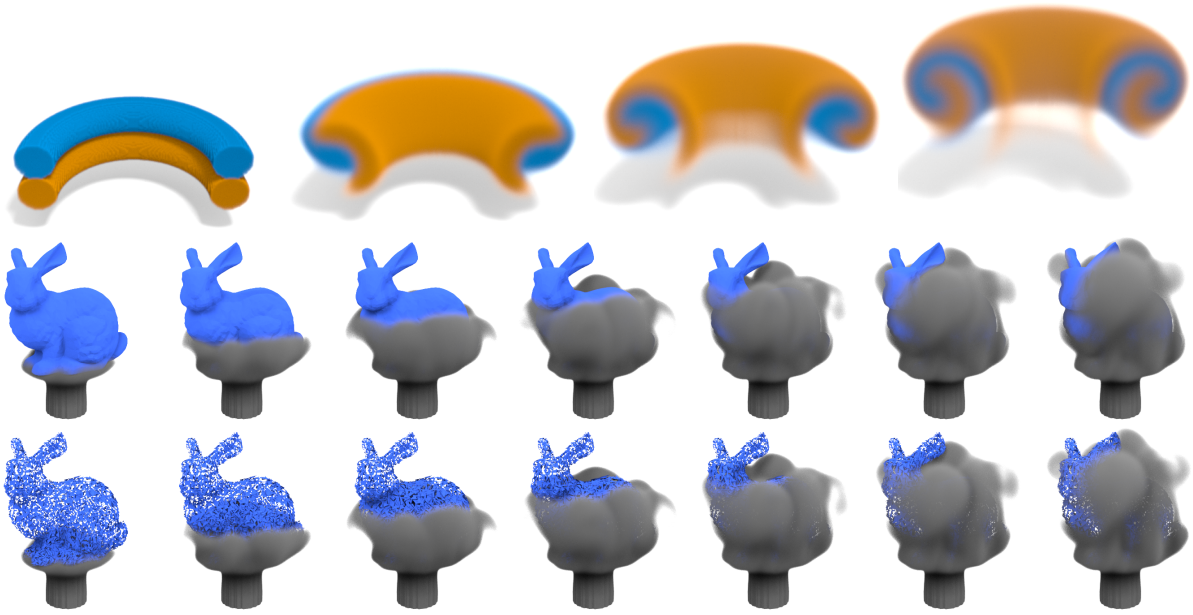


Figure 5.9: We simulate two leapfrogging vortex rings (top) with a cross section visualization to illustrate the interior flow. We passively advect constant density fields toward closed mesh (middle) and triangle soup Stanford bunnies (bottom). We render all simulations with Blender's [2024] principled volume shader.

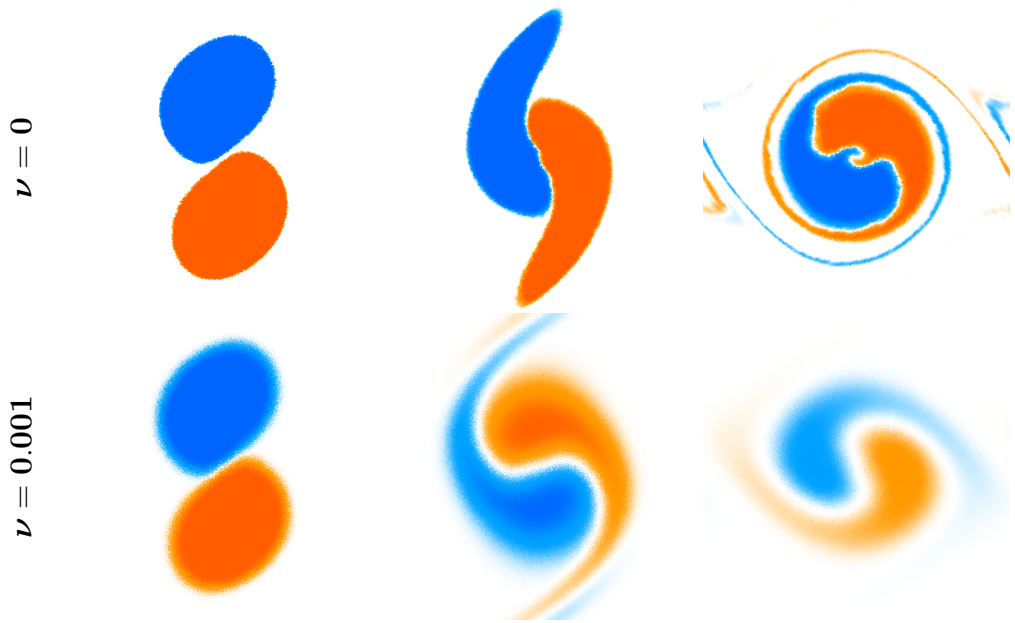


Figure 5.10: Viscous simulation. Left to right, $t = 0.2, 2, 5$ seconds. Inviscid simulation with $\nu = 0$ (top) and viscous simulation with $\nu = 0.001$; $n_{\mathrm{d}} = 4$. (bottom)

and better implementation of our Monte Carlo approach would reduce the performance gap, leaving the fundamental advantages of it as noted above.

Viscosity

Fig. 5.10 presents a 2D viscous simulation using our approach with a 512×512 uniform cache, nearest neighbor interpolation, importance sampling of the vorticity cache, $\Delta t = 0.05$, and $n_{\mathrm{mc}} = 1024$. Given that physical diffusion has an effect similar to interpolation, we found nearest neighbor (rather than bilinear) interpolation to be acceptable here.

Boundary Conditions

Fig. 5.2 presents various solid boundary configurations. All three scenes use a uniform cache of 512×512, $\Delta t = 0.05$, $n_{\mathrm{mc}} = 256$, and $\nu = 0$. As our method for free-slip boundary conditions relies on WoS, it benefits from some of the same attractive properties as the method of Sawhney and Crane [2020], e.g., flexibility in the choice of geometric representations, such as meshes, polygon soup, or even *unsigned* distance functions. In

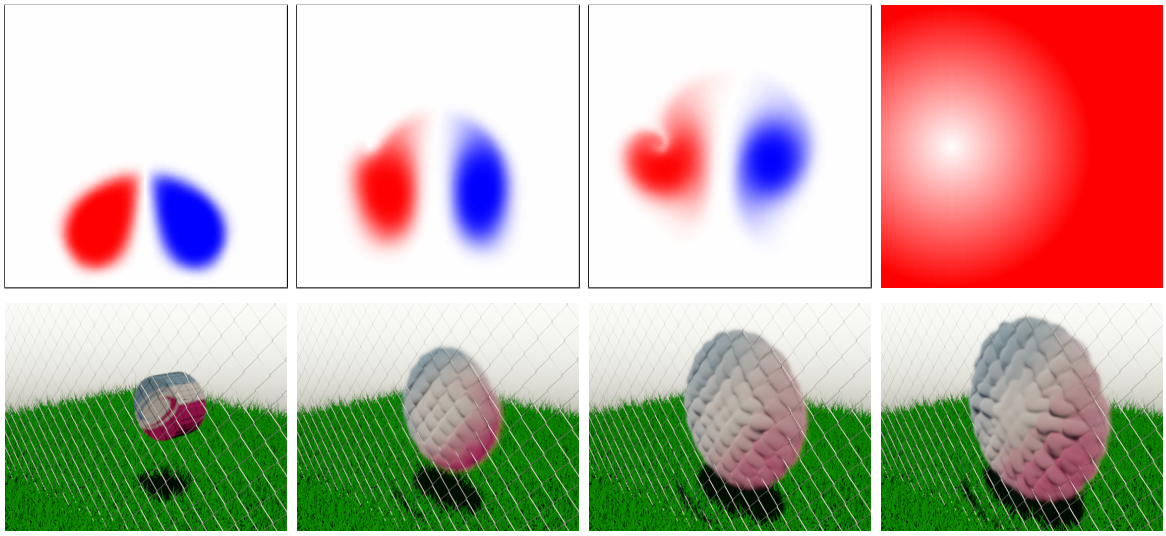


Figure 5.11: Subgrid-size obstacles. Our approach can take into account a tiny obstacle. A subgrid-sized square (the rightmost image visualizes its unsigned distance function) affects the fluid flow (top). In 3D, subgrid-width chain link fence wires affect the fluid flow (bottom).

Fig. 5.4, we illustrate our inflow/outflow boundary conditions. Since the integral of the stream function on the boundary is zero, the corresponding velocities yield zero net flux; that is, exactly as much matter enters the domain as exits.

Fig. 5.5 illustrates our moving boundary support. Here, we treat a constant translational velocity for simplicity, since this imposes a constant stream function on the boundary and simplifies computation. Obtaining an analytic formula for complex rigid bodies undergoing simple motion is a possibility for more advanced applications. Both the inflow and moving boundary applications were implemented as high-performance, real-time demos in the online GPU ShaderToy framework [Quilez and Jeremias 2013], demonstrating the suitability of our approach for parallel computation.

Fig. 5.9 (bottom) and Fig. 5.11 show that our method can accept complex geometry without the added effort of other methods [Azevedo et al. 2016; Hyde and Fedkiw 2019; Lyu et al. 2021], such as challenging tetrahedral mesh or cut-cell generation for velocity-based schemes or panel methods in vorticity schemes. As emphasized by Sawhney and Crane [2020], accurate conforming mesh generation for a complex obstacle can take many minutes or hours; however, as long as we have access to a method for fast evaluation of the distance field, our Monte Carlo-based methods can immediately be applied. In

the triangle soup boundary example in Fig. 5.9, we construct a BVH tree of triangles to evaluate distance to the closest boundary. Fig. 5.11 shows a case when the domain contains an obstacle whose size is smaller than the cache grid resolution (about 1/39 of a grid cell width for the 2D scene, and 1/3 of a cell width for the 3D scene), yet is naturally respected by the flow. A typical grid-based solver would either miss this geometry altogether, rely on conservative rasterization to nonphysically inflate the obstacle, or need to introduce an approximate drag model to influence the flow. Notably, the complex 3D fence topology of Fig. 5.11 exceeds the limits of what the simplified $\Psi = \mathbf{0}$ boundary condition is expected to accurately support, leading to somewhat more damped flow from one side to the other. Extending WoS to support the $\Psi = \nabla\phi$ boundary condition of Ando et al. [2015] may be one avenue to address this shortcoming.

Convergence

In the limit as Δt goes to zero and the number of samples approaches infinity, our method formally should converge to the correct solution (up to the accuracy limited by cache, when applicable). Fig. 5.8 qualitatively demonstrates that our method indeed gives a result that is consistent with existing techniques.

We opted to evaluate the convergence of our method using the steady-state inviscid Taylor-Green vortex flow [Taylor and Green 1937], for which a closed-form solution is known. Fig. 5.12 shows log-scale plots of the root mean square error (RMSE) against the number of Biot–Savart samples n_{mc}, number of cache cells $n_{\text{c}} = n_{\text{x}}^2$ and time step interval Δt. All errors were computed at the physical times $t = 0.25, 0.5, 1$ using importance sampling of the vorticity, bilinear interpolation, RK4 advection, without control variate. We also let $\Delta t = 2^{-6}$, $n_{\text{x}} = 2^{10}$ and $n_{\text{mc}} = 2^{10}$ whenever they are fixed. A well-known fact about Monte Carlo methods is that their error diminishes in inverse proportion to the square root of the number of samples, i.e. $\mathcal{O}(1/\sqrt{n_{\text{mc}}})$ [Robert and Casella 1999]. We confirmed this behavior in our experiment. Similarly, we observed $\mathcal{O}(1/\sqrt{n_{\text{c}}})$ which is reasonable given our use of bilinear interpolation. These trends are observed only when the other parameters are good enough to have negligible error contributions. Finally, we observed a convergence order of approximately $\mathcal{O}(\Delta t^{0.85})$. However, convergence is lost whenever Δt gets too small. This regime change is in line with the use of a semi-Lagrangian advection scheme [Xiu and Karniadakis 2001]. Intuitively, the error stops decreasing when the interpolation error begins to dominate the advection error. Since more steps are needed to reach the same physical time, the error increases as Δt gets smaller.

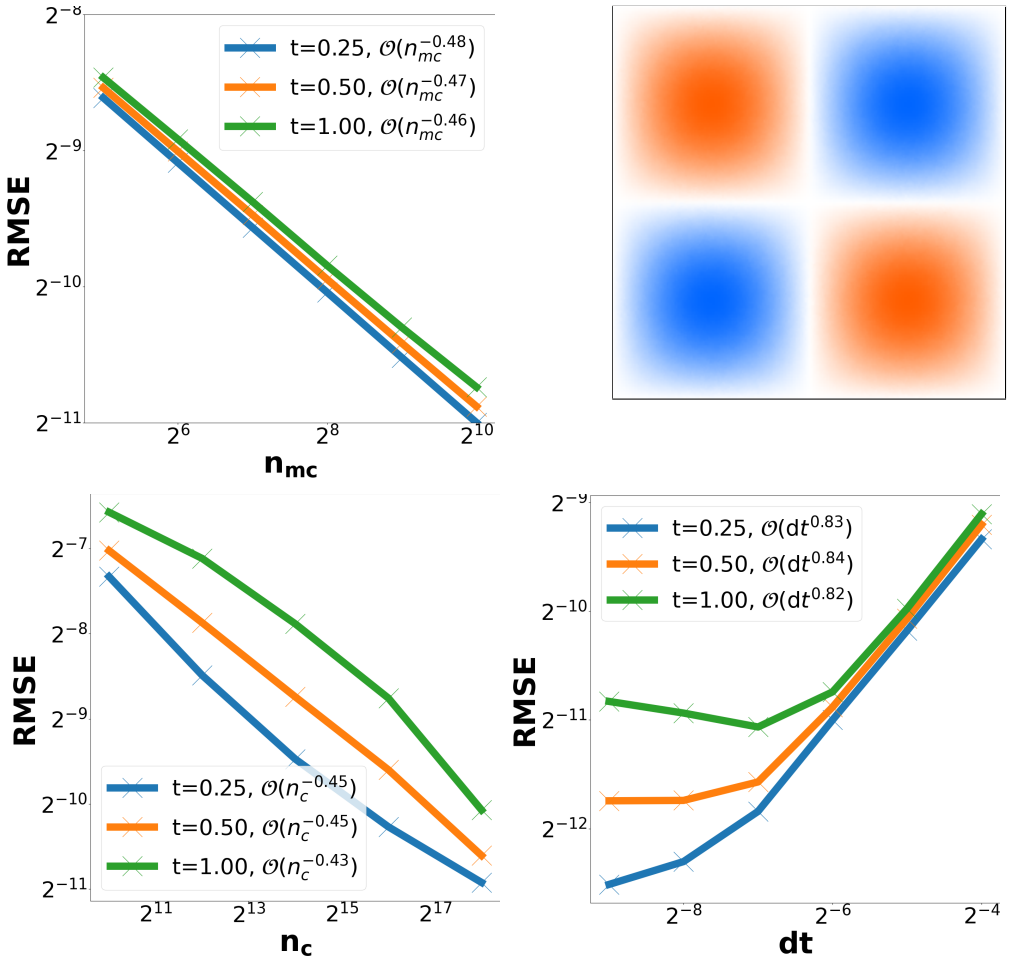


Figure 5.12: Convergence analysis. We compute the root mean squared error against the analytical solution $\omega(x, y) = \sin(x)\cos(y)$ for $x, y \in [-\pi, \pi]^2$ (top right). We show the convergence profile of different parameters at different physical times $t = 0.25, 0.5, 1$ seconds. We observe that our method does converge numerically to the true solution as the number of cell $n_c = n_x^2 \to \infty$ (bottom left), when the number of samples $n_{mc} \to \infty$ (top left) and when Δt is not too large or small (bottom right).

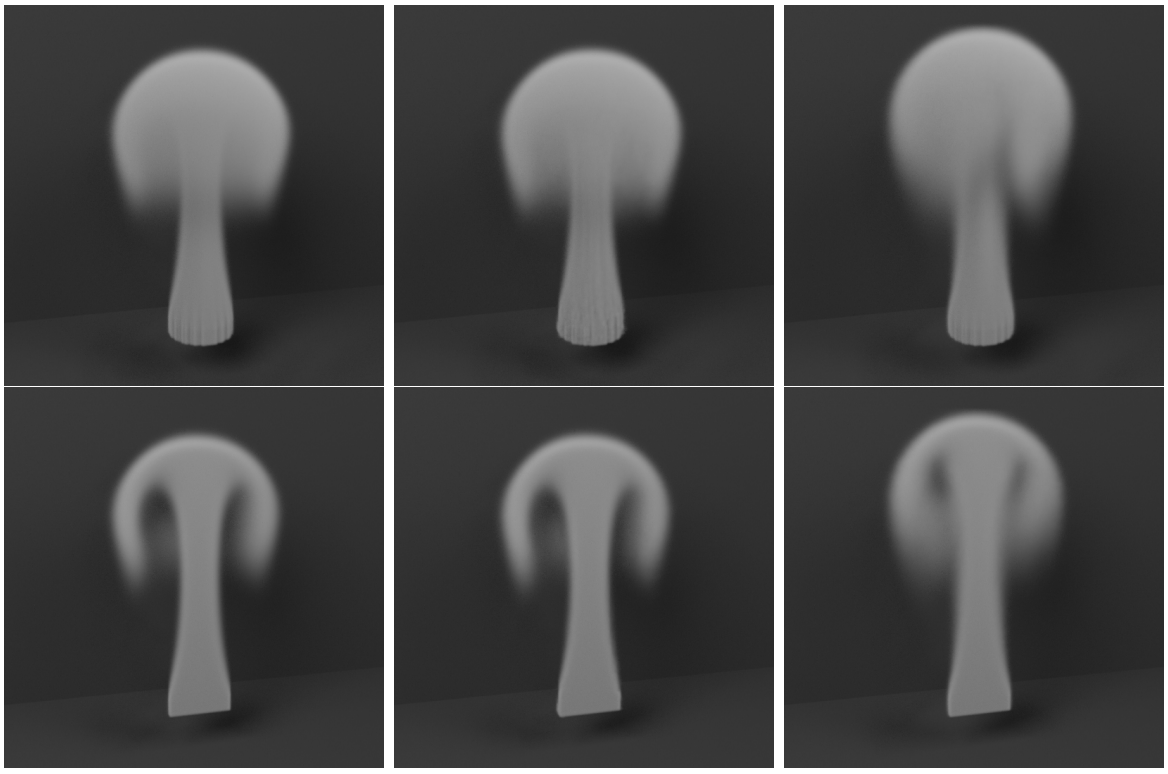
3D Simulations

We demonstrate our method in 3D, with and without obstacles (Fig. 5.9) using CUDA parallel GPU implementations. The leapfrogging simulation uses a uniform cache resolution of 256^3, trilinear interpolation, the control variate method enabled, $\Delta t = 0.1$, $n_{\text{mc}} = 128$, and $\nu = 0$. The initial velocity field is estimated with $n_{\text{mc}} = 16384$, importance sampling the initial vorticity field. The simulation time is roughly 3.5 seconds per step on a machine with two NVIDIA Tesla P100 GPUs at these settings. The Stanford bunny obstacle sceneswith slip boundaries uses a uniform cache resolution of 128^3, trilinear interpolation, importance sampling the Green's function, $\Delta t = 0.1$, 128 WoS paths with maximum 8 steps, 1024 samples for the volume integral evaluation (Eq. 14 in the paper by Sawhney and Crane [2020]), and $\nu = 0$. The simulation times are roughly 70 seconds for the triangle mesh boundary and 75 seconds for the triangle soup boundary per step on a machine with four NVIDIA Tesla V100 GPUs, using BVHs for closest distance computation. We believe this small difference in runtimes (an additional 5 seconds for triangle soup) is because of the additional cost for the closest distance evaluation.

We perform no low-level simulator optimization, focusing instead on the method's feasibility; GPU memory access optimizations and faster BVH traversal implementations can be explored. We also foresee and briefly discuss many interesting open avenues of work (Section 5.8) to improve performance, and scale to larger simulations drawing inspirations from the rendering community.

Control Variates

The method of control variates as described in Section 5.6 can greatly reduce the variance of simulation. Fig. 5.13 shows the effect of our control variate. All three simulations use a cache resolution of 128^3, trilinear interpolation, $\Delta t = 0.1$, and $\nu = 0$. The reference simulation (a) uses importance sampling according to the Biot–Savart kernel with $n_{\text{mc}} = 2048$. The simulation with control variate (b) uses the control variate strategy in Section 5.6 with $n_{\text{mc}} = 32$, with the initial velocity field estimated with $n_{\text{mc}} = 16384$ using importance sampling according to the initial magnitude of the vorticity field. The simulation without control variate (c) uses importance sampling according to the magnitude of the vorticity field with the same $n_{\text{mc}} = 32$. We can observe that even with as few as 32 samples per query point, the control variate approach produces a result very close to the reference while simply importance sampling the vorticity field leads to large deviations in the smoke motion due to the extensive noise in the estimated velocity field.



(a) reference (b) with control variate (c) w/o control variate

Figure 5.13: Control variate. When the number of samples used is small, the application of control variate (middle) produces the results closer to the reference simulation result (left) compared to the results without control variate (right). We advect a constant density at the bottom according to the velocity field simulated with our method, and render the results fully (top) and with a cross-section visualization (bottom).

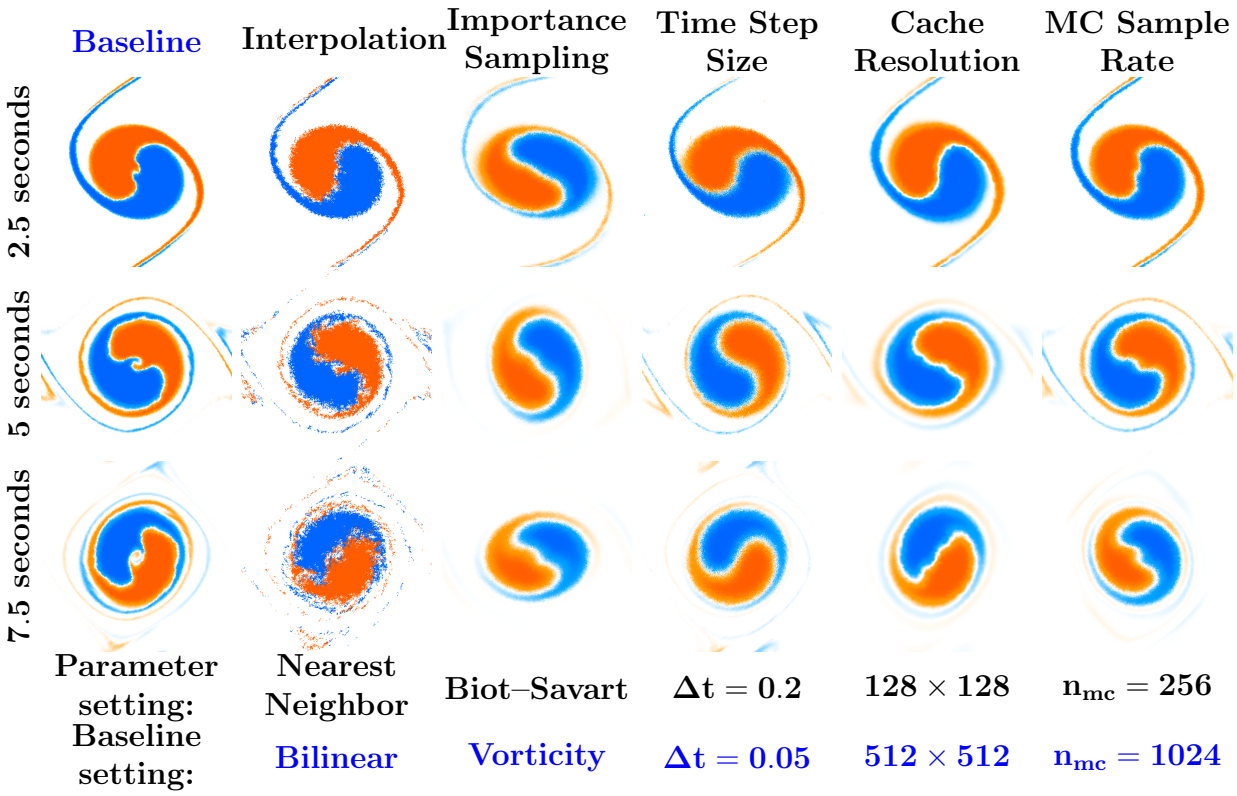


Figure 5.14: Parameter settings. Top to bottom, frame times are $t = 2.5, 5, 7.5$ seconds. Our baseline simulation (left) with high-quality parameter settings (in blue) uses a 512×512 uniform cache resolution, importance sampling of the vorticity field, bilinear interpolation, $\Delta t = 0.05$ seconds, $n_{\text{mc}} = 1024$ samples, and $\nu = 0$. Subsequent columns analyse the impact of adjusting exactly one parameter in favor of a faster but less accurate result. Left to right: baseline/high-quality setting, nearest neighbor interpolation, importance sampling the Biot–Savart kernel, larger time step, coarser cache, and lower Monte Carlo (MC) sampling rate.

Parameter Settings

To better understand the impact of the parameters of our solver on simulation output and performance, we illustrate the effects of the various parameters in Fig. 5.14. The first column is a reference simulation using our method with reasonably high-quality parameter settings: a uniform cache resolution of 512×512, importance sampling based on the vorticity field, bilinear interpolation, $\Delta t = 0.05$ seconds, $n_{\text{mc}} = 1024$ samples, and $\nu = 0$. All other columns modify a single parameter *in a manner that purposefully degrades simulation quality*. The results thus show how changing each parameter can degrade the solution.

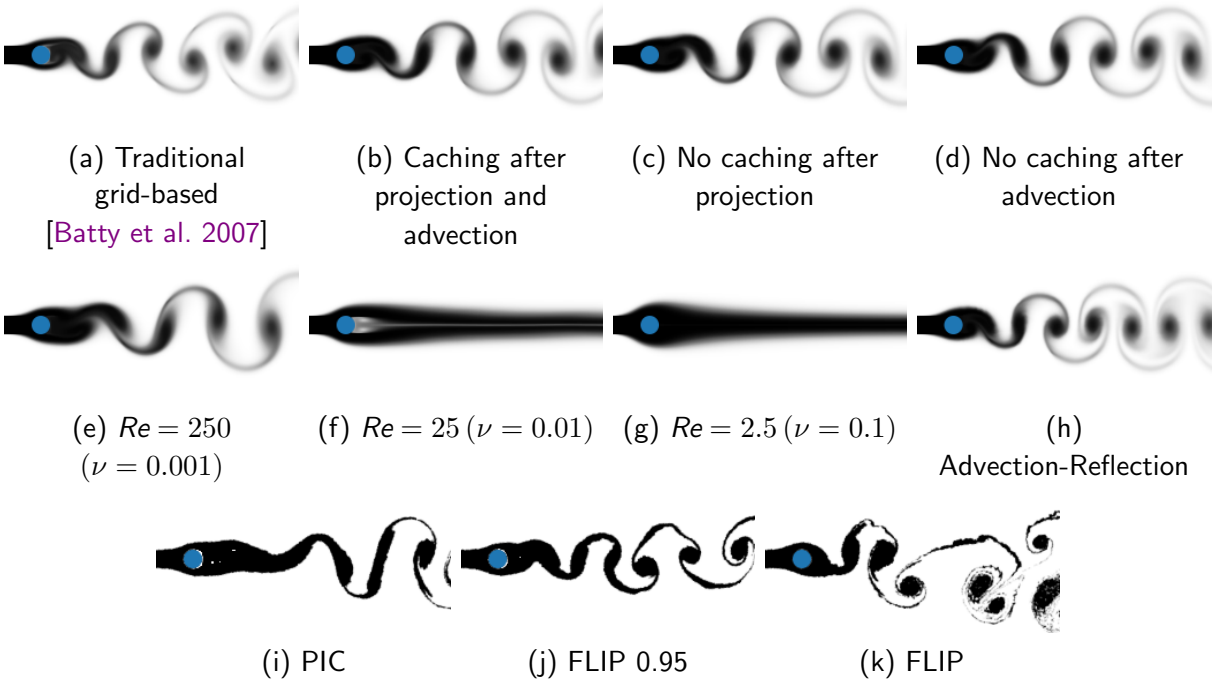
Due to the stochastic nature of Monte Carlo methods, nearest neighbor interpolation produces noisier (i.e., higher variance) results that compound over time. Since bilinear interpolation yields smoother results and is nearly free on, e.g., modern GPU architectures, it is the obvious choice. We also compare two non-trivial importance sampling PDFs to reduce the variance of the estimator and, as expected, importance sampling from the vorticity field yields sharper and more accurate results than sampling according to the Biot–Savart kernel (see Section 5.6) since the latter decreases slowly and omits information of the current simulation state. The discrepancy here is particularly strong when the vorticity is sparsely distributed across the domain. While our advection scheme is unconditionally stable, using time steps that are too large or a cache resolution that is too low results in inevitable loss of detail in high variation regions. Finally, since standard Monte Carlo error decreases at a rate of $\mathcal{O}(1/\sqrt{n_{\text{mc}}})$, the number of samples required for high quality results can be high; when we reduce the sample count by a factor of four, we note a significant loss of details.

5.7.2 Velocity-Based Method

We implemented the velocity-based method using CUDA for GPU parallelism and the NVIDIA OptiX ray tracing engine [Parker et al. 2010] via the OWL wrapper library [Wald 2023] for accelerated ray intersection queries for WoB.

Caching of Velocity Field

Our basic implementation uses a simple uniform grid structure to store the velocities at grid nodes at each time step. When we query a velocity from the cache, we bilinearly or trilinearly interpolate the values, and for points outside the cached domain, we take the



(a) Traditional grid-based [Batty et al. 2007]

(b) Caching after projection and advection

(c) No caching after projection

(d) No caching after advection

(e) $Re = 250$ ($\nu = 0.001$)

(f) $Re = 25$ ($\nu = 0.01$)

(g) $Re = 2.5$ ($\nu = 0.1$)

(h) Advection-Reflection

(i) PIC

(j) FLIP 0.95

(k) FLIP

Figure 5.15: Variants of our method with different configurations tested on the Kármán vortex street scene with resolution 128×256. Except for (a), all results are generated with our Monte Carlo method. (b) to (d) compare the caching alternatives, (e) to (g) demonstrate simulations with various Reynolds numbers Re, (h) shows the application of the advection-reflection method to yield reduced numerical dissipation, and (i) to (k) show the application of PIC/FLIP methods. Note that (i) to (k) have no density diffusion because they employ density advection by particles. Given their agreement with the result of the traditional grid-based method (a), we consider all variants of our method to produce qualitatively reasonable results.

velocity at the nearest cache point. As with classical grid-based solvers, interpolation- and advection-induced errors sometimes cause us to query values from points inside of solid obstacles; for this issue, we assign velocities inside solid obstacles to take velocities from the solid itself, giving no-slip behavior on the boundary. Alternatively, one could fill the velocities inside solid obstacles with some form of extrapolation to approximate free slip behavior, for example, but we use the first approach throughout this paper.

While we could cache the velocity field after each substep, the pointwise estimation capability of our approach lets us avoid some of the excessive velocity caching between substeps and reduce the associated errors. For example, we need no caching between the advection and the following projection step for inviscid simulation if we query the advected velocity values directly on the fly whenever the projection estimator needs such a pointwise input velocity. Alternatively, we can also design a method without velocity caching between the projection and the next advection, so the velocity field is always advected according to the pointwise divergence-free velocity field. We implemented these two options in addition to the option where we cache the resulting velocity field both after advection and after projection in Fig. 5.15 (b) to (d). Our experiments showed consistent results among the three options; further investigations are needed to understand when having fewer caching errors can make critical differences.

Viscous Fluids

We test our solver in the case of viscous fluids using the von Kármán vortex street scenario (Fig. 5.15 (e) to (g)). For this simulation, as we increase the viscosity coefficient, or equivalently, as we decrease the Reynolds number, the flow is expected to become less turbulent. We observe that our simulation qualitatively produces the expected type of flow for each Reynolds number configuration we tested. For all the results other than those in Fig. 5.15 (e) to (g), we disabled the diffusion step.

Advection-Reflection Solver

In contrast to the case of high viscosity, we may want to introduce less artificial viscosity to the solver. It is known that even without the diffusion step, grid-based solvers suffer from artificial viscosity in the solver because of the dissipation of energy in the advection and projection steps. To address this problem, Zehnder et al. [2018] recently introduced an advection-reflection solver, which still uses the same building blocks of advection and projection steps as the standard advection-projection solvers use, but in a more careful

way, to reduce artificial viscosity with the same number of projection steps. We adapted its second-order variant [Narain et al. 2019] to our Monte Carlo solver (Fig. 5.15 (h)), and we observe our method similarly benefits from the use of the advection-reflection method, producing more turbulent animation.

Monte Carlo PIC/FLIP

The numerical diffusion effect we have discussed above is partially due to the semi-Lagrangian advection scheme with some grid data interpolation that our method and many traditional methods employ. To address this problem in the context of classical grid-based solvers, the PIC/FLIP methods [Brackbill and Ruppel 1986; Harlow 1962; Zhu and Bridson 2005] have gained popularity. The idea of PIC/FLIP is that we handle the advection of velocity using particles by (forward) Lagrangian advection, transfer the velocity from particles to grid, perform projection on the grid, and update particle velocities by transferring grid data back to the particles. PIC and FLIP differ in their last grid-to-particle transfer step: PIC transfers the velocity itself, whereas FLIP transfers the velocity update. FLIP is expected to produce more turbulent and noisy results. One can also blend the two transfer methods. Our velocity-based Monte Carlo framework can utilize this idea straightforwardly (Fig. 5.15 (i) to (k)). In our version, at each step, we first perform the standard particle-to-grid transfer. Then, instead of advecting the particles using the projected grid velocities at the particles' locations via interpolation, we evaluate the velocity (or velocity update) from the grid data exactly at the positions required for particle advection. Exploiting the pointwise estimation capability in this way lets us avoid extra interpolation errors for this step.

Boundary Conditions

Fig. 5.7 shows that our method can correctly simulate the velocity field due to vortex pairs under different boundary conditions. In Fig. 5.6, we further demonstrate the application of our method to a domain with two disjoint obstacles. In such cases, vorticity-based methods often employ incorrect boundary conditions, resulting in the vorticity-based Monte Carlo method failing to accurately simulate the physics. Applying the modification proposed by Yin et al. [2023] for vorticity-based simulation to a Monte Carlo method is not straightforward, and we have not explored such methods in detail. Hence, this velocity-based method is currently the only Monte Carlo method that can produce correct simulation results in these scenarios. While the relative performance depends heavily on parameter choices, for the specific simulations in Fig. 5.7, ours took 101.0s per time step while the vorticity-based method took 87.6s, with a difference of about 15%.

Figure 5.16: Buoyancy and divergence control (resolution 256^2). We simulate a hot smoke plume rising toward a blue circular obstacle in 2D using the Boussinesq buoyancy model (left). We can also add a velocity sink to the scene, indicated with a red dot, causing the smoke to be sucked into the sink (right).

Buoyancy

In addition to the velocity itself, we can additionally simulate the advection and diffusion of the temperature field T and the smoke concentration field s in a manner similar to the velocity field to add buoyancy effects (Figures 5.1, 5.16, and 5.17). We use the Boussinesq approximation, which assumes the fluid density variation to be negligible, so that the buoyancy force can be computed using $\mathbf{f} = [\alpha s - \beta(T - T_{\text{ambient}})]\mathbf{g}$, where $\mathbf{g}$ is the gravitational constant vector, α and β are positive parameters, and T_{ambient} is the ambient temperature constant [Fedkiw et al. 2001; Foster and Metaxas 1997]. Naively incorporating buoyancy into the vorticity-based Monte Carlo formulation would require the curl of $\mathbf{f}$ [Park and Kim 2005], which would, in turn, require finite difference approximations with some additional errors and ruin the cache-structure-agnostic nature of pointwise estimators.

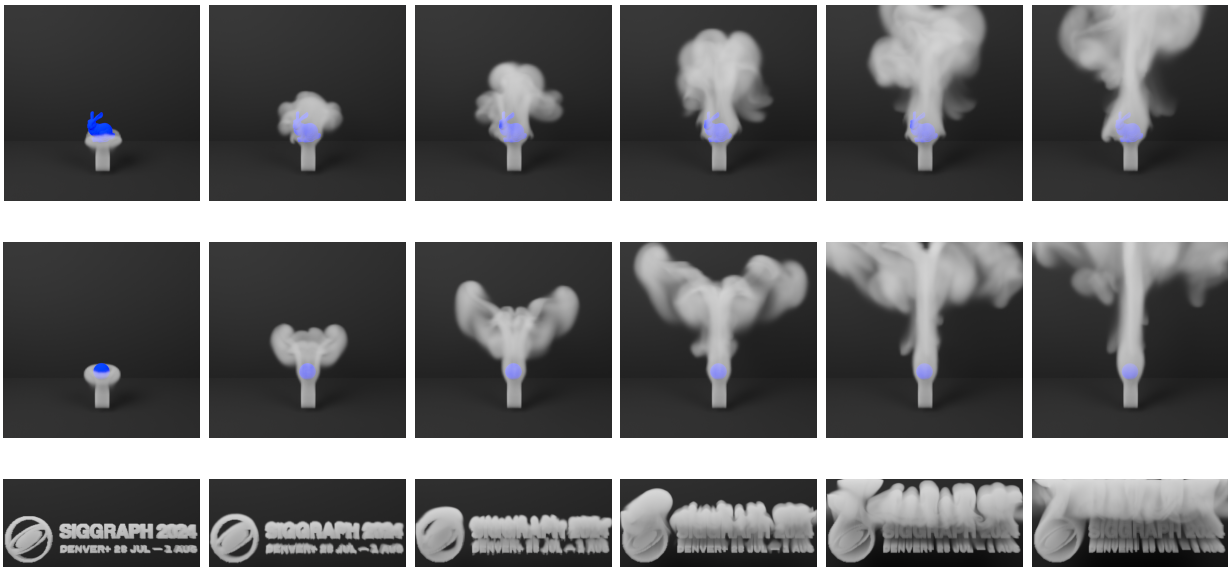


Figure 5.17: Buoyancy simulation in 3D. We simulate a smoke plume rising toward a bunny-shaped obstacle (top, resolution 128^3) and a sphere obstacle (middle, resolution 128^3) in 3D using the Boussinesq buoyancy model. We can also incorporate a more complicated smoke source shape as well (bottom, resolution $288 \times 128 \times 128$). We rendered the images with the principled volume shader in Blender [Blender Online Community 2024].

Divergence Control

As our formulation is based on the standard pressure solve framework, we can easily add velocity sinks and sources in our simulation for better artistic control (Fig. 5.16 (b)) or other expansion/contraction effects, by modifying the pressure solve step similar to the grid-based formulation [Feldman et al. 2003]. We add another term to the right-hand side of Eq. 5.17, which leads to an additional integral term in our formulation. Our implementation supports Dirac delta-type sinks and sources, and we sample the location of sinks and sources directly. It should also generalize straightforwardly to more general volumetric sinks and sources with an appropriate sampling technique. This divergence control technique is yet another advantage of adopting a velocity-based formulation, as it is impossible under vorticity formulations.

Performance

We measured the timing of our simulations in Fig. 5.15 using their GPU implementations against a basic single-thread CPU implementation of traditional grid-based fluids on a workstation with an Intel Xeon Silver 4316 CPU and an NVIDIA RTX A5000 GPU. Stable fluids with cut-cell boundary handling [Batty et al. 2007] (a) runs at 0.064 seconds per time step, while the fastest implementation of our solver that caches the velocity field after each substep (b) runs at 7.8 seconds per time step, two orders of magnitude slower. If we disable the VPL with this setup, the runtime increases to 109 seconds per time step. For the other caching choices, the runtime for the one without caching after projection (c) increases to 32.1 seconds per time step because we perform the projection at all points used in the RK3 advection, and the one without caching after advection (d) runs at 10.1 seconds per time step due to the increased number of evaluation points for the advection. In general, WoB for handling obstacles requires a large number of paths, and further efficiency improvements would be desirable to improve the method's practicality. The addition of diffusion steps in (e) to (g) resulted in an increase in runtime by 48% (e) to 84% (g) compared to their baseline counterpart implementation (c).

Error Analysis

We performed a convergence test for the projection step in the absence of obstacles. The estimator is unbounded in this case, and we can observe in Fig. 5.18 that the error decreases at the inverse square root rate, as we increase the number of samples to estimate the volume term and the area term, respectively. We also observe that using a simpler formulation or

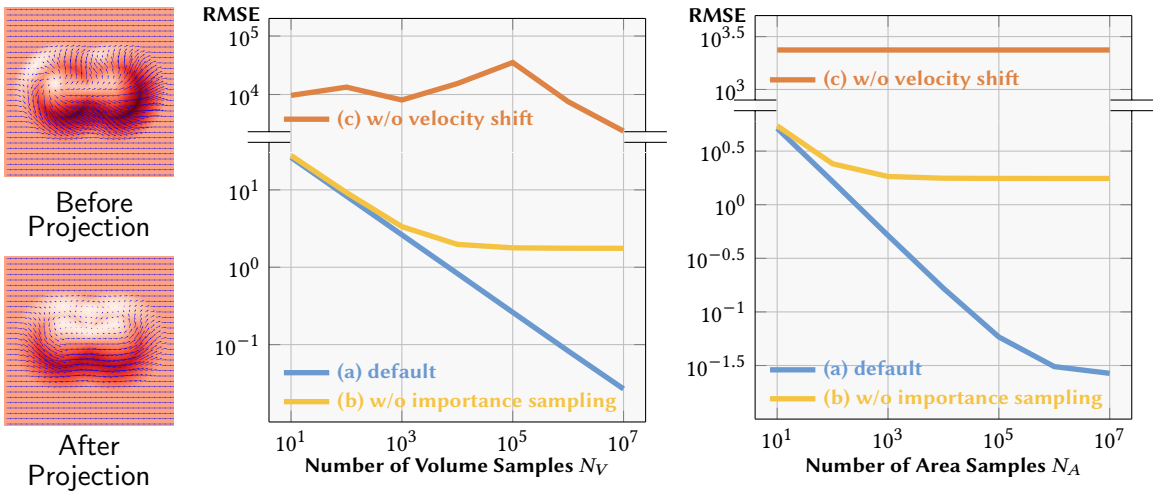


Figure 5.18: Projection error without solid boundaries. Our projection step projects out the divergent velocity mode from the velocity field, converting the top left field into the one on the bottom left (resolution 256^2). We visualize the velocity field using arrows, and we show its magnitude using the darkness of the red color as well. We measure the root mean squared errors against the analytical solution and plot them by altering the number of volume samples N_V with a fixed number of area samples $N_A = 10^7$ in the middle and by altering N_A with $N_V = 10^7$ on the right. We observe the inverse square root convergence rate in both cases with our formulation with the default sampling strategy (a) as described in Section 5.3.3, while the bias due to the volume term eventually dominates the error on the plot for N_A as we increase N_A. We also tested the estimator by (separately) replacing the volume term importance sampling according to the PDF proportional to $1/r^{(d-1)}$ with a uniform sampling (b), and disabling the global velocity shift described in Eq. 5.23 (c). Both of these alternatives either converge slower than the proposed method or fail to converge.

a simpler sampling technique makes the estimator converge more slowly or completely fail to converge. In Fig. 5.19, we compared our simulations with different numbers of samples. We observe that using low sample counts can blow up the simulation, and the noise of the velocity estimate is typically larger around the boundary. As we have discussed in Section 3.2, the variance of the Walk on Boundary method is particularly large for interior Neumann problems in non-convex domains, and Fig. 5.19 shows such a scene.

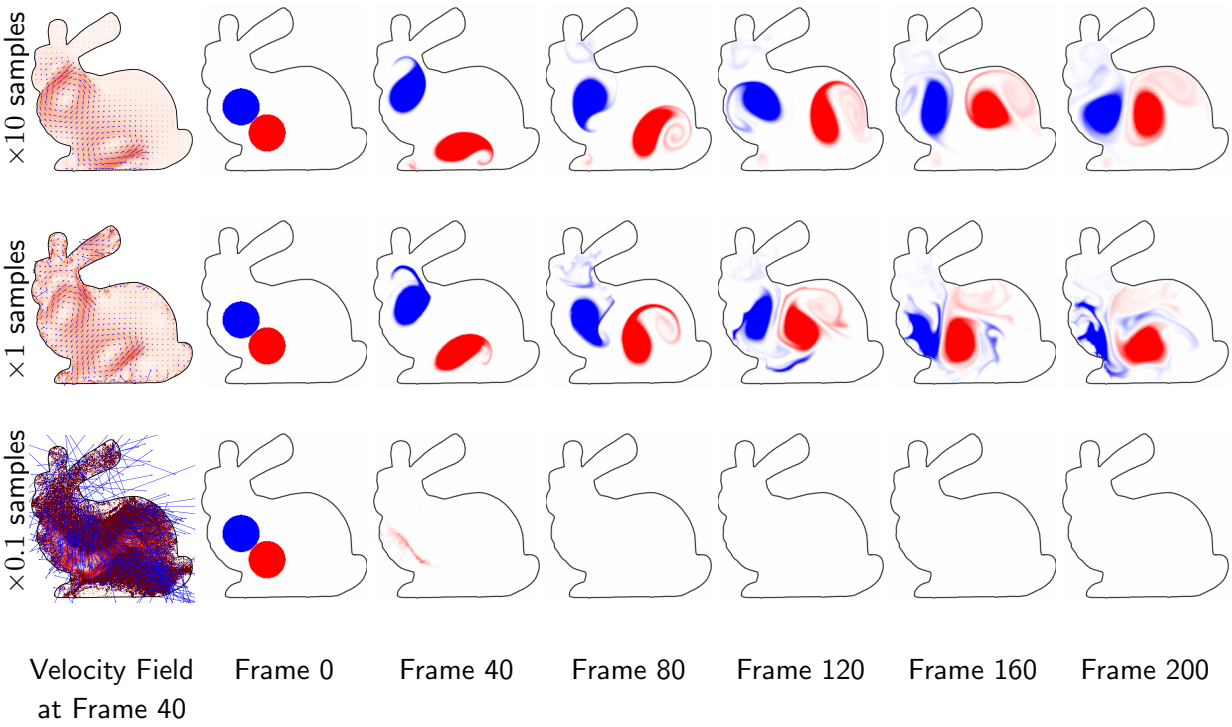


Figure 5.19: Flow in a 2D bounded non-convex domain (resolution 256^2). We simulate colored smoke advected by a velocity field induced by a vortex pair inside a bunny shape. On the left, the velocity field at the 40th frame is visualized. In these simulations, the number of samples used in each simulation is 10 times (top), 1 times (middle), and 0.1 times (bottom) the number of samples used in other 2D scenes in this paper. We observe that using a small sample count can lead to outright failure of the simulation (bottom). Using a modest number of samples (middle) in this scenario still exhibits large errors and jitter in the animation, likely due to larger velocity errors near the boundary and large globally correlated errors from the boundary value caching strategy. With the highest sample count (top), the simulation yields a smooth velocity field.

5.8 Discussion and Future Directions

Our work is the first step toward a new family of fluid solvers based on Monte Carlo methods. We demonstrated the potential of our Monte Carlo fluid solvers to generate high-quality and accurate results; however, there remains much to be explored, including a better understanding of how to best exploit the unique properties and new trade-offs of noise and computation time within fluid animation, among other general directions. Below, we discuss limitations and propose avenues of future work.

Efficiency

Our methods' computational speed is not yet comparable to traditional methods, as a relatively large number of samples is required to produce results with low enough error. Any variance reduction techniques for Monte Carlo PDE solvers, including the ones we listed in Section 2.3.5, should be applicable to improve the efficiency of our method. Moreover, employing alternative caching techniques or even alternatives to caching, as we discuss later, could improve the efficiency of our solver in practical scenarios.

Bias

Currently, our methods have three sources of bias: the first is due to cache interpolation error, the second is due to semi-Lagrangian advection, resulting in energy loss as time advances, and the third is due to the bias in Monte Carlo PDE solvers. The first source, e.g., the cache interpolation bias, is not inherent to our fundamental formulation. One way this could be resolved, without exponential cost, is through the use of a shared particle cache (and sampled locations for Monte Carlo integration) if one allows recursion to go back to the initial conditions. However, the second source, e.g., the advection bias, is intrinsic to our advection process, and solving this issue remains an open question. This is due to the fact that the nonlinearity of our advection scheme (Eq. 5.13) does not commute with the Monte Carlo estimator, much like naive transmittance estimation through ray marching is biased [Novák et al. 2018]. Further down the road, it would be interesting to explore whether the work of Misso et al. [2022] can be applied in our context to generate an unbiased estimator. The third source of bias is due to the errors in Monte Carlo PDE solvers for problems with solid obstacles. WoS is biased due to the epsilon shell path termination strategy, and the implementation of WoB we used is biased due to path truncation. Efficient, unbiased Monte Carlo PDE solvers are desirable.

Boundary conditions and Compressible Flows

Investigating a broader set of boundary conditions, such as no-slip boundaries for the vorticity-based formulation, mixed boundaries, and flux boundary conditions for multiphase flow, is another interesting direction.

With mixed boundaries, we can solve problems involving free surfaces [Lundgren and Koumoutsakos 1999] for liquid simulations. This requires an additional surface tracking mechanism, and it remains unclear what surface representation best suits Monte Carlo methods with pointwise evaluation capabilities. The recent development has been enabling the support of various boundary conditions by extending WoS [Sawhney and Miller et al. 2023; 2024b] or by WoB. However, compared to the solvers for pure Dirichlet problems (for the vorticity-based method) and pure Neumann problems (for the velocity-based method), these solvers are less efficient, and further improvements are awaited.

For the vorticity-based method, our free-slip boundary condition framework is only effective for a single closed and connected boundary; otherwise, one needs to know the difference in stream function isovalues between the separate boundaries. Similarly, a 3D vector potential is further complicated by its nontrivial null spaces; however, successfully generalizing to multiple disjoint objects could benefit from Monte Carlo methods' ability to handle far more complex geometries.

Support for flux boundary conditions in this context would enable the simulation of non-mixing multiphase flows, as demonstrated by Losasso et al. [2006]. Simonov [2008] and Ding and Stinchcombe [2025] considered Monte Carlo PDE solvers that incorporate such flux boundary conditions, and their methods may be applicable to our setting.

Apart from supporting different boundary conditions, extending our method to handle mixing multiphase flows and compressible flows, for example, in the simulation of combustion or explosions, would be a natural and interesting direction for future work. This would require solving Poisson equations with spatially varying coefficients. A promising starting point for this investigation is the Monte Carlo PDE solver proposed by Sawhney and Seyb et al. [2022], which could potentially be adapted to this context.

Advanced caching methods

We have presented how adopting a uniform grid cache avoids exponential cost and makes the method feasible in practice. For the velocity-based variant, we have also shown an application of PIC/FLIP Lagrangian particle representations. Though not associated with the Lagrangian formulation as in PIC/FLIP, particle caches are also used in rendering to

Figure 5.20: Fluid simulation results using a uniform cache (left) and an adaptive cache (middle) using one-twentieth memory footprint that of the uniform one. Slowly varying regions are sparsely cached while dense sampling occurs in regions exhibiting steep or discontinuous variations.

accelerate the computation of multiple bounces of light [Jarosz et al. 2008; Křivánek et al. 2005; Ward et al. 1988]. Since our equations have a similar structure to recursive equations in rendering, a careful allocation of cache particle points and reconstruction in our problem can be beneficial. Those prior works in rendering, however, would not be directly applicable to our method since fluid dynamics and light transport have different characteristics in terms of estimation errors (e.g., the split-sphere model in radiance caching [Ward et al. 1988] has no clear role in our framework).

A more classic alternative in the context of fluid animation is an adaptive grid cache, replacing the uniform grid with an adaptive tree structure to exploit sparsity. Due to our method's pointwise estimate feature, we can reduce the storage cost for caching *without* affecting the Monte Carlo estimator itself; other methods typically require modifications to their numerical algorithms, such as discrete gradient computation, for example. Figure 5.20 illustrates our adaptive cache prototype result. It exhibits a large reduction in memory, although this prototype does not yet offer a significant speed gain.

Alternatives to caching

Any implementations of our method with a cache result in additional bias beyond that introduced by time discretization. This issue motivates a desire to explore and devise alternative solutions for resolving the exponential cost of our base recursive formulation. This exponential cost, in its essence, is similar to the problem of having an exponential cost in path tracing [Kajiya 1986] if we were to split a path at each bounce (i.e., exponential

to the number of bounces). Path tracing avoids this exponential growth by tracing along only *one* direction per bounce. While we have empirically found that this approach is not applicable in our setting, some further study of Monte Carlo methods may avoid this exponential cost.

Quantum methods for numerical integration [Shimada and Hachisuka 2020] may be one potential avenue due to the exponential nature of computation via quantum bits. Russian roulette [Arvo and Kirk 1990], an unbiased method to terminate recursive Monte Carlo processes according to their expected contribution, can be applied to "early-out" fluid trajectories that contribute negligibly to the final dynamics. Similarly, path (resp. trajectory) splitting [Vorba and Křivánek 2016] can be used to adaptively refine our sampling.

Tighter coupling with rendering

Perhaps most excitingly, the similarities of our method to Monte Carlo light transport simulations suggest that a tighter integration between fluid and light transport simulations may prove fruitful. For example, since our fluid simulation generates errors that manifest as noise, and since possibly only a subset of the domain is rendered from any individual camera setting, one may be able to adapt the simulation sampling rate to account for the precision required by the rendering algorithm. Last-mile denoisers of rendered images [Chaitanya et al. 2017; Kalantari et al. 2015], further diversify this design space and trade-offs therein. For example, one might be able to design a denoiser that removes noise from *both* rendering and simulation (via our method) in a coupled manner.

Relatedly, heterogeneous volumetric rendering algorithms [Novák et al. 2018] build acceleration structures and variance reduction techniques to accelerate their convergence, and these structures can inform (and be informed) by Monte Carlo estimates of the underlying fluid dynamics. For example, one could imagine a hybrid approach in which a coarse grid-based simulation is used to guide the rendering process and high-resolution local re-simulations are performed on the fly using our approach when the renderer requires higher-resolution fluid density data.

Chapter 6

Monte Carlo PDE Solver for Surface PDEs

Methods to solve *surface* PDEs have become ubiquitous tools in computer graphics research and production. They are used for surface editing [Desbrun et al. 1999], texture synthesis [Turk 1991], surface fluid animation [Stam 2003], geodesic distance computation [Crane et al. 2013], and diffusion curves on surfaces [Bartsch et al. 2023; De Goes et al. 2022] today. A common approach to tackle these problems is to discretize the surface and solve a globally coupled linear system using discrete surface differential operators.

For *volumetric* PDEs, Monte Carlo methods have recently garnered significant attention in the graphics community due to their unique advantages over traditional discretization-based PDE solvers, including the ability to estimate solution values in a pointwise, spatial discretization-free manner. One such method is the *Walk on Spheres* (WoS) method [Muller 1956] introduced to graphics by Sawhney and Crane [2020]. They primarily focused on the (constant coefficient) Poisson and screened Poisson equations in a volumetric domain, and follow-up work, including the solvers we have discussed in the previous chapters, likewise focuses on volumetric problems. In this chapter, we consider instead the problem of solving PDEs on surfaces.

Sawhney and Seyb et al. [2022] proposed an extension of WoS to support second-order linear elliptic PDEs with spatially varying coefficients, and as one application, they demonstrated a method to solve the Laplace equation on a 2D surface embedded in 3D. However, this approach requires that the conformal parametrization of the surface be readily available, limiting the method's applicability.

We propose a simpler generalization of WoS for surface PDEs, the *Projected Walk on*

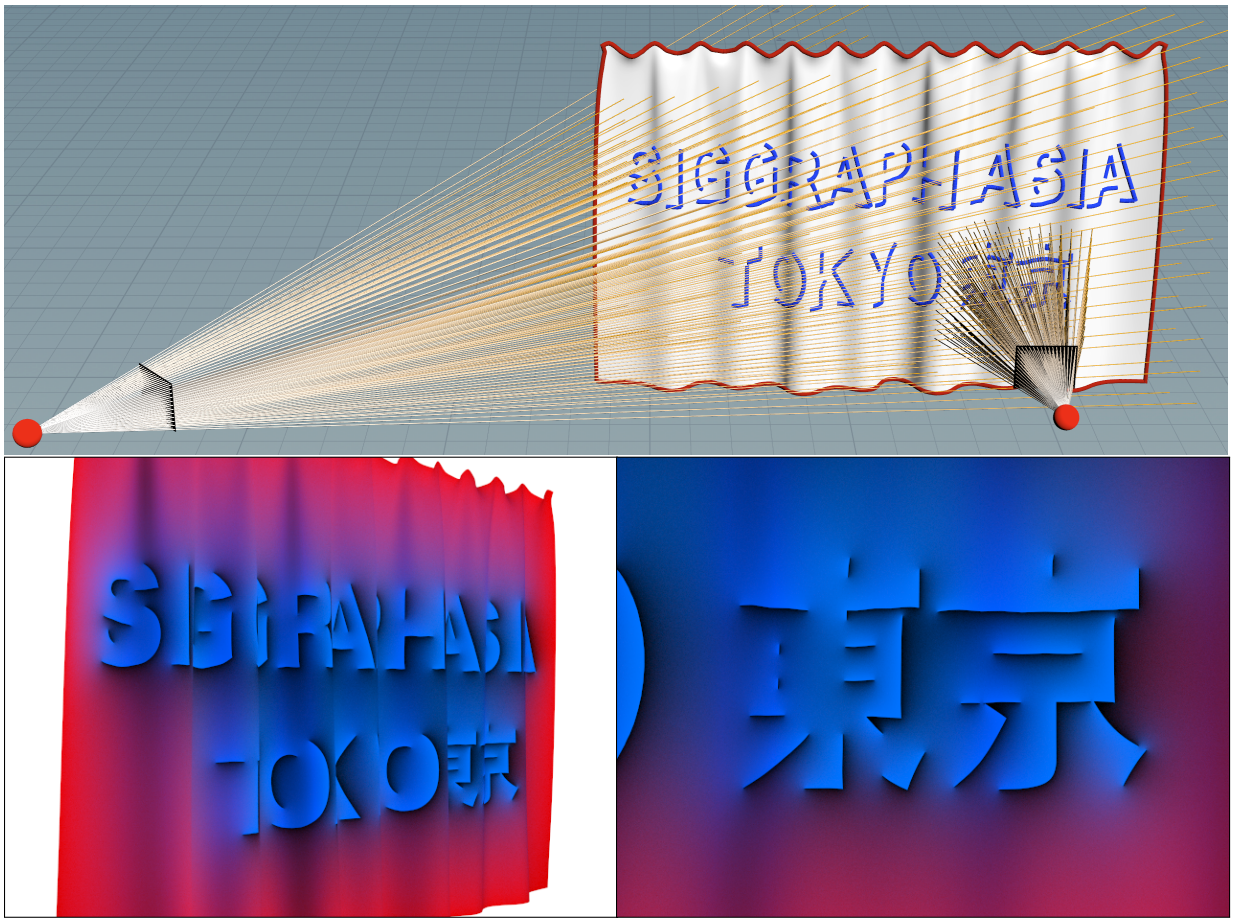


Figure 6.1: View-dependent diffusion curves with PWoS. Using our method, we solve the Laplace equation on a curved surface in a view-dependent manner. The pointwise and discretization-free nature of PWoS allows for the evaluation of the colors only at visible points where the object color is required by a shading algorithm with stochastic pixel-filtering.

Spheres (PWoS) method, which only assumes the availability of a closest surface point query and an unoriented surface normal direction query. PWoS supports Dirichlet boundary conditions and inherits the advantages of WoS: PWoS is a pointwise, discretization-free Monte Carlo method. Since our method does not require the meshing of the surface, it is particularly advantageous over traditional approaches, such as the finite element method, when the computation can be localized and when the surface is given as an implicit representation, such as a signed distance function. The resulting solution is also free of mesh-dependent discretization artifacts, such as from linear interpolation, as we show in Fig. 6.1. Compared to WoS, which performs walks on spheres inside the domain, PWoS performs walks on spheres inside a Cartesian embedding neighborhood domain around the surface. After each step of the walk, it *projects* the sampled point on the sphere onto the surface. We motivate this simple modification to the original WoS through its connection to the *closest point method* (CPM) [März and Macdonald 2012; Ruuth and Merriman 2008].

Furthermore, inspired by the mean value filtering method for WoS by Bakbouk and Peers [2023], we design a mean value filtering method with a discrete basis function to allow more efficient estimation of solutions when the surface is represented as a polygonal mesh or a point cloud. To confirm PWoS's accuracy, we perform convergence studies of the method applied to the surface Poisson and screened Poisson equations. Finally, we demonstrate its use in several representative graphics applications, including diffusion curves, geodesic distance computation, and wave propagation animation.

6.1 Surface PDEs and Closest Point Extension

Consider the Poisson equation defined on a surface $\mathcal{S}$ embedded in $\mathbb{R}^d$ such that $\dim(\mathcal{S}) < d$:

$$\nabla^2_{\mathcal{S}} u_{\mathcal{S}}(\mathbf{y}) = f_{\mathcal{S}}(\mathbf{y}), \quad \mathbf{y} \in \mathcal{S}, \tag{6.1}$$

where $\nabla^2_{\mathcal{S}}$ denotes the Laplace-Beltrami operator. For convenience, we will use the word surface to refer to any nonzero codimension object in $\mathbb{R}^d$, including mixed-codimension objects. One typically solves such a surface PDE by discretizing the differential operator and solving a sparse linear system. For triangle meshes one can use the cotangent Laplacian [MacNeal 1949]; for other surface representations, a corresponding discrete Laplacian must be defined [Belkin et al. 2009; Bunge and Botsch 2023; Sharp and Crane 2020]. The closest point method [Ruuth and Merriman 2008] instead addresses the surface PDE (Eq. 6.1) in a more general way by changing the domain to an embedding space, which is a Cartesian neighborhood surrounding the original surface. CPM then solves an *embedding PDE* defined on the embedding space, whose solution when restricted to points $\mathbf{y} \in \mathcal{S}$ is

the solution $u_{\mathcal{S}}(\mathbf{y})$ to the original surface PDE. We briefly summarize CPM theory and refer readers to work by King et al. [2024] for an in-depth review of CPM.

We first assume that $\mathcal{S}$ is smooth and define the closest point query to the surface, for $\mathbf{x} \in \mathbb{R}^d$, as

$$\mathrm{cp}_{\mathcal{S}}(\mathbf{x}) = \underset{\mathbf{y}\in\mathcal{S}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{y}\|_2. \tag{6.2}$$

In general, the mapping $\mathrm{cp}_{\mathcal{S}}(\mathbf{x})$ may not be unique: there may exist more than one closest point for a given $\mathbf{x}$. We define the neighborhood $\mathcal{N}(\mathcal{S})$ where the closest point function is unique as $\mathcal{N}(\mathcal{S}) = \left\{\mathbf{x} \in \mathbb{R}^d \;\middle|\; \|\mathbf{x} - \mathrm{cp}_{\mathcal{S}}(\mathbf{x})\|_2 < \mathrm{LFS}\left(\mathrm{cp}_{\mathcal{S}}(\mathbf{x})\right)\right\}$, where $\mathrm{LFS}(\mathbf{y})$ is the local feature size at point $\mathbf{y} \in \mathcal{S}$ defined as the minimum Euclidean distance from $\mathbf{y}$ to the medial axis $\mathrm{med}(\mathcal{S})$ [Amenta and Bern 1999]. The medial axis $\mathrm{med}(\mathcal{S})$ is defined as the set of points in $\mathbb{R}^d$ where there is more than one closest point. Note that when $\mathcal{S}$ is a watertight surface, the definition of the medial axis that we use contains both the interior part that is bounded by $\mathcal{S}$ and the exterior part that lies outside the bounded domain.

Within the neighborhood $\mathcal{N}(\mathcal{S})$, or a subset of it, surface differential operators can be replaced by Cartesian differential operators with *closest point extensions* (see [März and Macdonald 2012; Ruuth and Merriman 2008]). The closest point extension operator E extends surface functions onto $\mathcal{N}(\mathcal{S})$ to be constant in the normal direction of $\mathcal{S}$ and is defined as $Eu_{\mathcal{S}}(\mathbf{x}) = u_{\mathcal{S}}(\mathrm{cp}_{\mathcal{S}}(\mathbf{x}))$. For functions $u \in \mathcal{N}(\mathcal{S})$ the extension E acts on the restriction of u to the surface, i.e., $Eu = E(u|_{\mathcal{S}})$. The Laplace-Beltrami operator in Eq. 6.1 is equivalent to the following:

$$\nabla^2_{\mathcal{S}} u_{\mathcal{S}}(\mathbf{y}) = \nabla^2[Eu_{\mathcal{S}}](\mathbf{y}), \quad \mathbf{y} \in \mathcal{S}. \tag{6.3}$$

To define the embedding PDE on $\mathcal{N}(\mathcal{S})$, we also extend $f_{\mathcal{S}}$ as $f(\mathbf{x}) = Ef_{\mathcal{S}}(\mathbf{x})$. The equation $\nabla^2[Eu_{\mathcal{S}}](\mathbf{x}) = f(\mathbf{x})$, for $\mathbf{x} \in \mathcal{N}(\mathcal{S})$, is ill-posed because f is constant in the normal direction of $\mathcal{S}$ but $\nabla^2[Eu_{\mathcal{S}}]$ is not guaranteed to be. Therefore, the embedding PDE for Eq. 6.1 becomes

$$\nabla^2[Eu_{\mathcal{S}}](\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x}), \quad \mathbf{x} \in \mathcal{N}(\mathcal{S}), \tag{6.4}$$

where $g(\mathbf{x})$ is a function that compensates for $\nabla^2[Eu_{\mathcal{S}}]$ not being constant in the normal direction of $\mathcal{S}$. The function $g(\mathbf{x})$ is nonzero for $\mathbf{x} \in \mathcal{N}(\mathcal{S}) \setminus \mathcal{S}$ and $g|_{\mathcal{S}} = 0$ to ensure Eq. 6.4 is consistent with the surface PDE (Eq. 6.1) on $\mathcal{S}$. Any function g with these conditions has the form $g(\mathbf{x}) = \gamma(v(\mathbf{x}) - Ev(\mathbf{x}))$, where $\gamma \in \mathbb{R}$ and $\gamma \neq 0$.

The Macdonald-Brandman-Ruuth approach (see [Chen and Macdonald 2015, Section 2.3]) takes $v|_{\mathcal{S}} = u_{\mathcal{S}}$ to allow Eq. 6.4 to be written as an equation in one unknown, $v(\mathbf{x})$, since $Eu_{\mathcal{S}} = Ev$ (but importantly $v \neq Ev$ except on $\mathcal{S}$). We instead do not restrict $v|_{\mathcal{S}}$

to be $u_{\mathcal{S}}$ and interpret $g(\mathbf{x})$ as a modification to the source term $f(\mathbf{x})$, then solve for the unknown solution $u(\mathbf{x}) = Eu_{\mathcal{S}}(\mathbf{x})$ in Eq. 6.4. As proven by von Glehn et al. [2013, Section 3.2], $u(\mathbf{x}) = Eu(\mathbf{x})$ since $u(\mathbf{x})$ is the extension of a surface function. The property that $u(\mathbf{x}) = Eu(\mathbf{x}) = u(\text{cp}_{\mathcal{S}}(\mathbf{x}))$ allows our projection step during the walk in PWoS, detailed in Section 6.2.

We show through our numerical examples that taking $g(\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathcal{N}(\mathcal{S})$, provides qualitatively correct results for graphics applications and quantitatively convergent results in most examples in Section 6.4. However, the choice of $g(\mathbf{x}) = 0$ causes Eq. 6.4 to be ill-posed as discussed above, and we observe some bias in some convergence studies in Section 6.4.1 when f is complex. Interesting future work would involve constructing a more accurate $g(\mathbf{x})$ function to improve convergence.

In the traditional CPM, one solves the embedding PDE inside a narrow tubular subset of $\mathcal{N}(\mathcal{S})$ that is within a constant distance to $\mathcal{S}$. For the typical grid-based variant, the tubular subset is spatially discretized with a grid of uniform spacing. Interpolation and finite differences are applied on the grid to approximate the closest point extension and the spatial Cartesian differential operators, respectively, and then the resulting linear system is solved. Other variants of CPM [Cheung et al. 2015; Petras and Ruuth 2016; Piret 2012] also require some discretization within $\mathcal{N}(\mathcal{S})$. Thus, while traditional CPM is agnostic to the specific surface representation, it still discretizes the embedding space and solves a globally coupled system. Moreover, imposing exterior or interior boundary conditions requires tedious grid operations [King et al. 2024]. By contrast, we develop a spatial discretization-free, pointwise Monte Carlo estimator for surface PDEs by incorporating the closest point extension concept into WoS.

When there are Dirichlet boundaries $\mathcal{C} \subset \mathcal{S}$, on which the solution $u_{\mathcal{S}}$ is given, one can geometrically extend the boundary itself out into the embedding space in the normal directions, assigning it the same boundary value in accordance with the closest point extension. Note that such boundaries need not coincide with the geometric boundaries of the surface itself. In the context of grid-based CPM, King et al. [2024] discuss how to impose such boundary conditions by duplicating degrees of freedom near the extended boundary. In our work, we devise a method that uses only the closest point function $\text{cp}_{\mathcal{C}}(\mathbf{x})$ to the (pre-extension) boundary $\mathcal{C}$, without the need to construct the extended boundary geometry or perform any complex duplication of degrees of freedom.

6.2 Method

Input

While our algorithm is generalizable to other configurations, we describe our method for the case when $\mathcal{S}$ is embedded in $\mathbb{R}^3$ and $\dim(\mathcal{S}) = 1, 2$. Recall that we use the word surface to refer both to "surfaces" with $\dim(\mathcal{S}) = 1$ (curves) as well as $\dim(\mathcal{S}) = 2$ surfaces. We also allow mixed codimension where parts of the surface have $\dim(\mathcal{S}) = 1$ and the rest of the surface has $\dim(\mathcal{S}) = 2$. We assume that we can query the closest point function $\text{cp}_{\mathcal{S}}(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^3$. Additionally, for surfaces with $\dim(\mathcal{S}) = 2$, we assume that we can query the unoriented normal direction of the surface $\mathbf{n}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{S}$. For surfaces with $\dim(\mathcal{S}) = 1$, we assume that we can query the tangential direction of the surface $\mathbf{t}(\mathbf{x})$ for $\mathbf{x} \in \mathcal{S}$. These assumptions are valid for common surface representations, including, but not limited to, polygonal meshes, oriented point clouds, and implicit functions. The theory discussed in Section 6.1 is based on the assumption that $\mathcal{S}$ is smooth; in practice, we observe that applying our technique on discretized surfaces with sharp features behaves well as we show in Section 6.4.1. Additionally, we assume the Dirichlet boundaries $\mathcal{C}$ have a lower dimension than the dimension of $\mathcal{S}$ and support the closest point query $\text{cp}_{\mathcal{C}}$. When solving a two-sided boundary value problem for boundaries with $\dim(\mathcal{C}) = 1$, we also assume that we can query the tangent direction of $\mathcal{C}$.

Overview

The core idea of our method is to apply the WoS recursive relationship within $\mathcal{N}(\mathcal{S})$ while utilizing the closest point extension constraint that $u(\mathbf{x}) = u(\text{cp}_{\mathcal{S}}(\mathbf{x}))$. To do so, we modify the walk process to use spheres contained within $\mathcal{N}(\mathcal{S})$ and to *project* the walk position at each recursion step, as illustrated in Fig. 6.2.

The problem we solve is the embedding PDE $\nabla^2 u(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x})$ within $\mathcal{N}(\mathcal{S})$. The Monte Carlo estimate of WoS (Eq. 2.18) holds by assuming $g(\mathbf{x}) = 0$ because the embedding PDE is defined with the Cartesian differential operator. To estimate the surface PDE's solution at point $\mathbf{x} \in \mathcal{S}$, we consider a 3D ball centered at $\mathbf{x}$ and fully contained inside $\mathcal{N}(\mathcal{S})$. Theoretically, it should be the largest ball fully contained inside $\mathcal{N}(\mathcal{S})$ that does not cross the extended Dirichlet boundaries $\mathcal{C}$, to minimize the number of steps needed to reach the boundary. We determine the radius of such a ball by taking the minimum of a conservative (under-)estimate of the local feature size at point $\mathbf{x}$ (Section 6.2.1) and the distance to the (extended) Dirichlet boundaries (Section 6.2.2). In Eq. 2.18, the Monte Carlo estimate of the solution on the sphere, $\widehat{u}(\mathbf{y})$, needs to be evaluated at point $\mathbf{y}$, which does not lie on $\mathcal{S}$

Algorithm 2: Projected Walk on Spheres

Input: surface $\mathcal{S}$, boundary $\mathcal{C}$, evaluation point $\mathbf{x} \in \mathcal{S}$, sample walk count N_P, volume sample count N_V, tolerance ϵ

Function `EstimateSolution`($\mathcal{S}$, $\mathcal{C}$, N_P, N_V, $\mathbf{x}$, ϵ)**:**

- $\mathcal{M} \leftarrow$ `medialAxisPointCloud`($\mathcal{S}$) // Section 6.2.1
- $\widehat{u}_{\text{sum}} \leftarrow 0$
- **for** $n \leftarrow 1$ *to* N_P **do**
  - $\widehat{u} \leftarrow$ `RecursiveEstimate`($\mathcal{S}$, $\mathcal{M}$, $\mathcal{C}$, N_V, $\mathbf{x}$, ϵ)
  - $\widehat{u}_{\text{sum}} \leftarrow \widehat{u}_{\text{sum}} + \widehat{u}$
- **end**
- **return** $\widehat{u}_{\text{sum}}/N_P$

Function `RecursiveEstimate`($\mathcal{S}$, $\mathcal{M}$, $\mathcal{C}$, N_V, $\mathbf{x}$, ϵ)**:**

- $\delta \leftarrow$ `distanceToBoundary`($\mathcal{S}$, $\mathcal{M}$, $\mathcal{C}$, $\mathbf{x}$) // Section 6.2.2
- **if** $\delta < \epsilon$ **then**
  - **return** $u(\text{cp}_{\mathcal{C}}(\mathbf{x}))$
- $l \leftarrow$ `localFeatureSize`($\mathcal{S}$, $\mathcal{M}$, $\mathbf{x}$) // Section 6.2.1
- $R \leftarrow \min(l, \delta)$
- $\mathbf{y} \leftarrow$ `uniformSphereSample(center=`$\mathbf{x}$ `, radius=`R`)`
- $\widehat{u}_{\text{sphere}} \leftarrow$ `RecursiveEstimate`($\mathcal{S}$, med, $\mathcal{C}$, N_V, $\text{cp}_{\mathcal{S}}(\mathbf{y})$, ϵ)
- $\{\mathbf{z}_1, ...\mathbf{z}_{N_V}\} \leftarrow$ `ballSample(center=`$\mathbf{x}$ `, radius=`R`)`
- $\widehat{u}_{\text{ball}} \leftarrow \frac{1}{N_V} \sum_{i=1}^{N_V} \frac{G(\mathbf{x},\mathbf{z}_i) f(\text{cp}_{\mathcal{S}}(\mathbf{z}_i))}{p(\mathbf{z}_i)}$
- **return** $\widehat{u}_{\text{sphere}} + \widehat{u}_{\text{ball}}$

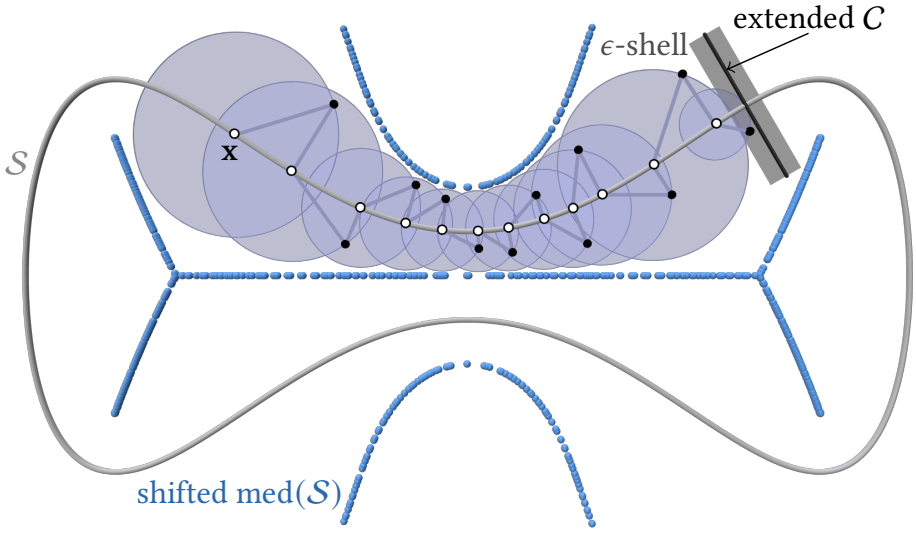


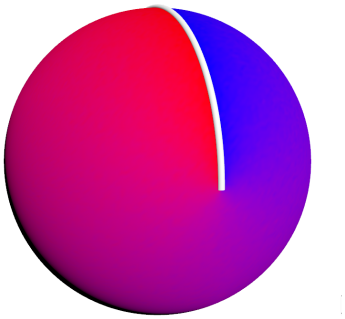
Figure 6.2: A PWoS path for the Laplace equation on a gray 1D (curve) surface embedded in 2D space, starting from $\mathbf{x}$ and terminating at the extended Dirichlet boundary $\mathcal{C}$.

in general. The closest point extension constraint of u provides a convenient relationship here: the embedding PDE's solution at point $\mathbf{y}$ should coincide with the surface PDE's solution at the projected point, $\text{cp}_{\mathcal{S}}(\mathbf{y})$. We can therefore project the point $\mathbf{y}$ onto $\mathcal{S}$ at the end of each recursion step, hence $\widehat{u}(\mathbf{y}) = \widehat{u}(\text{cp}_{\mathcal{S}}(\mathbf{y}))$. After this projection at the end of each step, we continue the recursion. The source term similarly uses the closest point projection for the closest point extension, replacing the recursive relationship of WoS (Eq. 2.18) with

$$\widehat{u}(\mathbf{x}) = \widehat{u}(\text{cp}_{\mathcal{S}}(\mathbf{y})) + \frac{1}{N_V} \sum_{i=1}^{N_V} \frac{G_R(\mathbf{x}, \mathbf{z}_i) f(\text{cp}_{\mathcal{S}}(\mathbf{z}_i))}{p(\mathbf{z}_i)}. \tag{6.5}$$

We choose $p(\mathbf{z}_i) \propto 1/\|\mathbf{x} - \mathbf{z}_i\|_2$ in our implementation.

Analogous to the original WoS method, we terminate the recursion when the point $\mathbf{x}$ falls within a distance ϵ of the (extended) Dirichlet boundary by taking the boundary value $u(\text{cp}_{\mathcal{S}}(\mathbf{x}))$. We provide pseudocode for an instance of our algorithm in algorithm 2, where we highlight the difference between our proposed method and WoS. We also provide a visualization of a potential path of our algorithm when $\mathcal{S}$ is embedded in $\mathbb{R}^2$ in Fig. 6.2. Notably, PWoS is a generalization of the WoS algorithm: when $\dim(\mathcal{S}) = d$ (i.e., the codimension-zero case), the local feature size is infinite, the distance to the Dirichlet boundary $\mathcal{C}$ can be computed with the closest point query $\text{cp}_{\mathcal{C}}$, and the last closest point projection of $\mathbf{y}$ has no



Local Feature Size	Avg. Step Count
0.99	31.1398
0.5	47.84
0.25	128.981
0.125	462.781
0.0625	1818.73

Figure 6.3: Average number of steps required with different conservative local feature size estimates. While any positive value smaller than 1 is valid for this setup, using a local feature size estimate that is too small leads to excessively long walks.

effect since $\text{cp}_{\mathcal{S}}(\mathbf{y}) = \mathbf{y}$. When $\dim(\mathcal{S}) < d$, in addition to the closest point projection at the end of each recursion step, our algorithm utilizes two nontrivial steps: the local feature size estimation using a medial axis point cloud and the computation of the distance to the (extended) Dirichlet boundary. We discuss these in the following two subsections.

6.2.1 Local Feature Size Estimation

To determine the radius of the sphere centered at $\mathbf{x} \in \mathcal{S}$ that is fully contained inside $\mathcal{N}(\mathcal{S})$ at each recursion step of the walk, we need a conservative lower bound estimate for the local feature size at $\mathbf{x}$. One naive approach would be to use a small enough positive constant value for all $\mathbf{x} \in \mathcal{S}$, similar to the grid-based CPM [Ruuth and Merriman 2008]. This is a valid strategy, but it would often yield a sphere radius smaller than necessary, requiring more recursion steps for the walks to reach a Dirichlet boundary and making the method inefficient. Fig. 6.3 illustrates a result of our algorithm on a unit sphere using different (artificial) local feature size estimates. The analytical local feature size of this surface is 1 everywhere. Although using any smaller value would still give consistent results, it significantly increases the average step count for the walks. For more complex shapes, one cannot compute the analytical local feature size in general and using a small constant value in its place is inefficient. This issue motivates our need for a better local feature size estimate.

To estimate the local feature size, we compute a point cloud approximation of the medial axis and estimate the local feature size as the distance from any query point $\mathbf{x} \in \mathcal{S}$ to its nearest point in the medial axis point cloud. One could use any local feature size estimation algorithm and/or medial axis extraction algorithm (see e.g., [Tagliasacchi et al.

2016]), such as one that outputs or uses the medial axis' connectivity. Since such methods are often comparatively costly, we employed a simple point cloud-based method, which we describe below.

Medial Ball Extraction

We first densely scatter points $\mathbf{x}_i$ inside a ball in $\mathbb{R}^3$ having a radius equal to half the length of the diagonal of the bounding box of $\mathcal{S}$, so the entire surface is fully enclosed. For each point $\mathbf{x}_i$, we use the closest point query $\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_i)$ to compute its two opposing normal directions at $\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_i)$. Specifically, we normalize the vector $\mathbf{x}_i - \mathrm{cp}_{\mathcal{S}}(\mathbf{x}_i)$ to get the first direction and invert its direction to get the second one. We then use the method of Ma et al. [2012] to extract a point cloud that represents the medial axis, as follows. For each side of the surface (i.e., each normal), we start with a large sphere tangent to $\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_i)$, whose center necessarily lies on the normal ray. The initial radius of the ball is set to the length of the diagonal of the bounding box of $\mathcal{S}$. Then, we iteratively shrink the size of the sphere, moving its center to maintain tangency at the surface point $\mathrm{cp}_{\mathcal{S}}(\mathbf{x}_i)$, until the closest surface point from the center of the sphere does not change. This algorithm gives *medial balls*, i.e., balls with their centers on the medial axis. As the number of initial scattered points increases, the extracted point cloud balls tend to cover the entire medial axis. While Ma et al. [2012] assumed that the surface is represented as an oriented point cloud, we observed that the algorithm works well with other surface representations by adjusting its termination criteria. See the paper by Ma et al. [2012] and our implementation provided with the published paper [Sugimoto et al. 2024b] for details.

Scale Axis Pruning

Directly using the medial ball centers as the medial axis point cloud does not work well in general when $\mathcal{S}$ contains any noise or artificial sharp corners introduced by the discretization of a smooth surface. We therefore prefer to estimate (only) the *stable* part of the medial axis, which is not affected by any small perturbation of $\mathcal{S}$. A common solution is, therefore, to *prune* unstable components of the medial axis, which itself remains an active research topic [Tagliasacchi et al. 2016]. We take inspiration from the scale axis transform (SAT) [Giesen et al. 2009; Miklos et al. 2010], but design an alternative since SAT considers topology information of the medial axis, which is unnecessary for local feature size estimation. Our alternative is simpler and faster since topology information is omitted. We first scale all the medial balls by a constant factor $s > 1$. Then, for any pair of medial balls $B_{r_1}(\mathbf{x}_1)$ and $B_{r_2}(\mathbf{x}_2)$, we remove the smaller ball from the set of medial balls if it is

fully contained in the other. That is, if $s \cdot r_1 < s \cdot r_2 + \|\mathbf{x}_1 - \mathbf{x}_2\|_2$, we remove the ball $B_{r_1}(\mathbf{x}_1)$.

Note that some of the medial balls may have a very large radius before pruning. For example, an exterior medial ball for a surface of a convex shape would have an infinite radius in theory (but our algorithm returns at most the length of the diagonal of the bounding box of $\mathcal{S}$). When such large medial balls are used in our pruning algorithm, they can easily (and undesirably) remove important, stable parts of the medial axis. The original SAT approach was applied only to the interior medial axis of closed surfaces. Therefore, this issue was not observed since the interior medial ball radii are always bounded and proportional to the size of local features of $\mathcal{S}$. To address this problem, we consider each pair of tangent balls generated at the same surface point, and replace the larger one with a tangent ball having the radius of the smaller one. In other words, before we prune the medial axis, we shift the medial ball centers of the larger medial ball in the pair and shrink its radius. In Fig. 6.2, we visualize the medial ball centers after this shifting operation.

After this pruning, the set of medial ball centers gives the medial axis point cloud we use to estimate the local feature size. Adjusting the scaling parameter s allows us to control the pruning strength. Unless otherwise stated, we use the value $s = 1.15$ for our results.

Conservative and Nonzero Local Feature Size

As the medial axis is represented as a point cloud and the nearest point distance may give a larger value than the actual local feature size, we multiply by a small constant (0.9 in our implementation) to ensure a conservative estimate of the local feature size. When there are sharp corners in the geometry, the analytical local feature size is zero, and the walk will become stuck. To prevent this problem, when the estimated local feature size is smaller than a positive constant threshold λ, we return λ as the local feature size estimate instead. This process essentially rounds sharp corners with rounding radius λ. The uniform grid size adopted in grid-based CPM has a similar effect. We do not observe any significant error due to this rounding, as we show in Section 6.4.1.

6.2.2 Distance to Extended Dirichlet Boundaries

Dirichlet boundaries $\mathcal{C}$ are extended in the normal direction of $\mathcal{S}$ and the solution in the embedding space on this extended boundary is determined by the closest point extension of Dirichlet values on $\mathcal{C}$. Therefore, we need to compute the minimum distance to the extended Dirichlet boundaries, and limit the sphere radius in PWoS further if it is less

than the local feature size. To determine the distance to the extended Dirichlet boundary from point $\mathbf{x} \in \mathcal{S}$, we first find the closest point that lies on the boundary before the extension, $\text{cp}_{\mathcal{C}}(\mathbf{x})$. The subset of the extended boundary that is extended from $\text{cp}_{\mathcal{C}}(\mathbf{x})$ takes the shape of a line segment when $\dim(\mathcal{S}) = 2$ and a disk when $\dim(\mathcal{S}) = 1$. We set the line segment's half-length or the disk radius to the local feature size at $\text{cp}_{\mathcal{C}}(\mathbf{x})$ using the algorithm in Section 6.2.1. We can compute the distance from the point $\mathbf{x} \in \mathcal{S}$ to this line segment or disk without explicitly constructing the extended boundary geometry. When $\dim(\mathcal{S}) = 2$, the distance δ to the extended boundary is given by

$$\begin{aligned} \mathbf{r} &= \text{cp}_{\mathcal{C}}(\mathbf{x}) - \mathbf{x}, \\ \delta &= \|\mathbf{r} - \text{clamp}(\mathbf{r} \cdot \mathbf{n}, -l, l) \cdot \mathbf{n}\|_2, \end{aligned} \tag{6.6}$$

where $\mathbf{n}$ and l are the surface normal and the local feature size estimate at $\text{cp}_{\mathcal{C}}(\mathbf{x})$, respectively. When $\dim(\mathcal{S}) = 1$, the normal direction is not uniquely defined, so we instead use a similar method based on the surface tangent $\mathbf{t}$:

$$\begin{aligned} \mathbf{q} &= \mathbf{r} - (\mathbf{r} \cdot \mathbf{t})\mathbf{t}, \\ \delta &= \begin{cases} |\mathbf{r} \cdot \mathbf{t}|, & \text{if } \|\mathbf{q}\|_2 < l, \\ \|\mathbf{r} - l \cdot (\mathbf{q}/\|\mathbf{q}\|_2)\|_2, & \text{otherwise.} \end{cases} \end{aligned} \tag{6.7}$$

6.2.3 Generalizations

So far, we have focused on estimating the solution to the surface Poisson equation in its standard form. However, we can naturally extend the method to support more advanced scenarios, building on prior work in related areas. One of the simplest extensions is the screened Poisson equation, which appears in graphics applications such as the heat method for geodesic distance computation [Crane et al. 2013]. Additionally, many graphics applications, including the heat method and the projection step in fluid simulation [Foster and Metaxas 1996; Stam 1999], lead to Poisson equations with source terms given by the divergence of a vector field: $f_{\mathcal{S}} = \nabla_{\mathcal{S}} \cdot \mathbf{h}_{\mathcal{S}}$, where $\mathbf{h}_{\mathcal{S}}$ is a vector field defined on the surface. These applications also often require estimating the gradient of the solution, rather than the solution itself. With grid-based CPM, the differential operators defined in the embedding Cartesian domain can be used to solve such problems [Auer et al. 2012; King et al. 2024]. For our PWoS, however, we do not use any embedding grid structure, and we do not assume any specific surface representation, so we cannot use such discrete differential operators. Instead, we address this challenge by drawing insights from the Monte Carlo PDE solver literature.

Screened Poisson Equation

For volumetric PDEs, there exists a generalization of WoS to the screened Poisson equation $\nabla^2 u - \sigma u = f$, where σ is a positive constant (Section 2.3.2). The embedding PDE for the screened Poisson equation is constructed similarly to Eq. 6.4 using closest point extensions [Chen and Macdonald 2015, Section 2.3], so the projection step in our algorithm is justified. Therefore, similar to WoS, we replace the original integral equation (Eq. 2.17) used to derive the recursive formulation of PWoS with its screened counterpart (Eq. 2.19). Since PWoS is a generalization of WoS, we can directly apply the same sampling strategy, including the Russian Roulette path termination outlined in Section 2.3.2.

Gradient Estimation

To estimate the gradient with PWoS, we can similarly replace the first step of recursion of PWoS by the one for gradient estimation in WoS (Section 2.3.3), because the solution's gradient is zero in the surface normal directions due to the closest point extension constraint $u(\mathbf{x}) = u(\mathrm{cp}_S(\mathbf{x}))$. The surface gradient of the solution to a surface PDE does not have a normal component, but the estimated solution may have a nonzero normal component before convergence. Thus, to improve the estimate, we set the normal component(s) of the estimated gradient to zero as a post-processing step.

Divergence Source Term

To solve a problem with a divergence of a vector field as the source term, in Section 5.3, we proposed to use integration by parts to convert the volume integral arising from the source term to a form that does not explicitly depend on the divergence of the vector field. This was done in the context of the Walk on Boundary method, and we adapt this approach to (Projected) WoS here. To estimate the solution with the source term $f = \nabla \cdot \mathbf{h}$, the

volume term converts to

$$\begin{aligned}
&\int_{B_R(\mathbf{x})} f(\mathbf{z}) G(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z} \\
&= \int_{B_R(\mathbf{x})} \left(\nabla_{\mathbf{z}} \cdot \mathbf{h}(\mathbf{z})\right) G(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z}, \\
&= \int_{\partial B_R(\mathbf{x})} \mathbf{h}(\mathbf{z}) \cdot \mathbf{n}(\mathbf{z}) \, G(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z} - \int_{B_R(\mathbf{x})} \mathbf{h}(\mathbf{z}) \cdot \nabla_{\mathbf{z}} G(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z}, \\
&= - \int_{B_R(\mathbf{x})} \mathbf{h}(\mathbf{z}) \cdot \nabla_{\mathbf{z}} G(\mathbf{x}, \mathbf{z}) \, \mathrm{d}\mathbf{z},
\end{aligned} \tag{6.8}$$

and we evaluate the last integral instead, which does not require the explicit evaluation of the divergence of $\mathbf{h}$. We generate the samples to estimate the converted volume integral with $p(\mathbf{z}) \propto 1/\|\mathbf{x} - \mathbf{z}\|_2^2$, so the singularity of $\nabla_{\mathbf{z}} G$ cancels out.

We demonstrate an application of the divergence source term and gradient estimation support for the geodesic distance computation in Section 6.4.2.

6.3 Mean Value Filtering with Discrete Basis

The method we described in Section 6.2 is suitable for evaluating the solution at a few evaluation points independently. To improve the efficiency of the method for many evaluation points (i.e., mesh vertices), it is often desirable to utilize the spatial consistency of the solution. For WoS, a few such methods have been proposed, but those approaches are not trivially applicable to our setup, where we have additional closest point extension constraints between the surface and embedding PDEs. Specifically, the boundary value caching approach [Miller and Sawhney et al. 2023] is based on a boundary integral equation defined in the Cartesian coordinates and is not applicable to surface PDEs. Adaptation of reverse WoS [Qi et al. 2022] or mean value caching [Bakbouk and Peers 2023] to our setting would require a mapping of a PDF on the surface to a PDF on the spheres used in our walk process, and we found it difficult to design such an algorithm for general surfaces.

Inspired by the filtering method of Bakbouk and Peers [2023], we designed a filtering method that uses discrete basis functions defined over a surface represented as a polygonal mesh or point cloud. When we apply Eq. 6.5 at an evaluation point, if $\widehat{u}(\mathrm{cp}_{\mathcal{S}}(\mathbf{y}))$ is a precomputed Monte Carlo estimate, we do not need to continue the recursion and can simply use the precomputed estimate in its place. In general, we do not have the estimate $\widehat{u}(\mathrm{cp}_{\mathcal{S}}(\mathbf{y}))$ available at $\mathrm{cp}_{\mathcal{S}}(\mathbf{y})$. Thus, we get the estimate by interpolating the estimate

of solutions already computed at a discrete set of points by accepting the bias due to interpolation. For example, for a triangle mesh, we use barycentric coordinates as the interpolation basis. This can be interpreted as a PDE-aware smoothing filter of the solution when we consider this process as a weighted average of the solution estimates at nearby evaluation points.

To estimate the solution at all mesh vertices or all points in a point cloud, we first compute an approximate solution to the problem with a low sample count using the method described in Section 6.2 and apply this filtering step to get an updated estimate. We can also precompute the filtering weights per evaluation point first and apply the same filter a few times to achieve an even smoother solution without too much additional cost. We can similarly design a gradient estimation filter based on Section 6.2.3. While this filtering approach utilizes a discrete basis defined over the surface, we do not use any explicit discrete differential operators and do not need to solve a linear system, which is in contrast to the traditional methods based on discrete Laplacians.

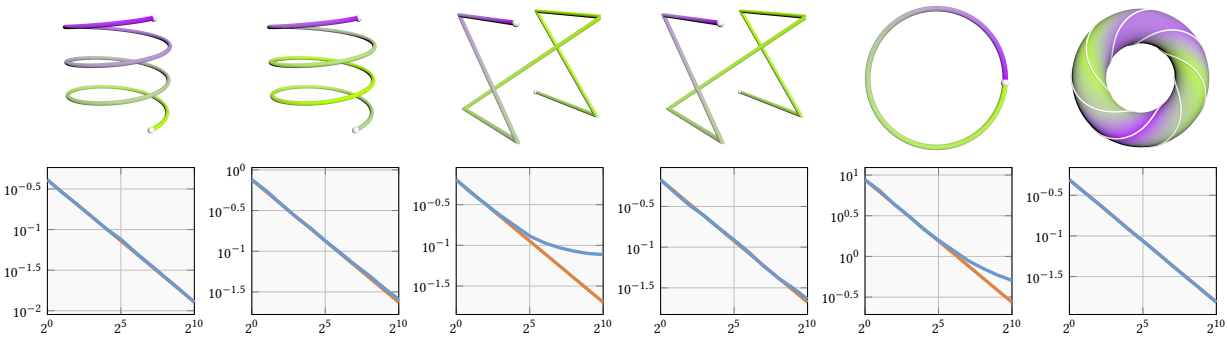
The method of Bakbouk and Peers [2023] similarly designed a smoothing filter for volumetric PDEs. Their method can keep the estimate unbiased by assuming that the evaluation points are sampled with a known PDF and that the original estimates are unbiased, but our method introduces bias due to the use of a discrete basis. We nevertheless observe that, within a reasonable runtime budget, our biased filtering method can reduce the error compared to the PWoS algorithm without filtering. We leave the development of an unbiased variant to future work.

6.4 Results

We implemented PWoS in Houdini 20.0 [Side Effects Software Inc. 2024] with CPU parallelization, without GPU acceleration, using its built-in closest point queries.

6.4.1 Error Convergence

We conducted error convergence studies of our method using discretized surfaces. In each scene in Fig. 6.4, we change the number of sample paths N_P to plot the root mean squared error measured against the analytical or reference solution. We do not apply the mean value filtering method in these studies. The scene setup includes Laplace, Poisson, and screened Poisson equations, and both smooth surfaces and surfaces with sharp corners. In all scenes, we use $\epsilon = 0.001$ and $N_V = 32$ except for Laplace equations, for which we use



(a) Helix Ends Laplace

(b) Helix Ends Poisson

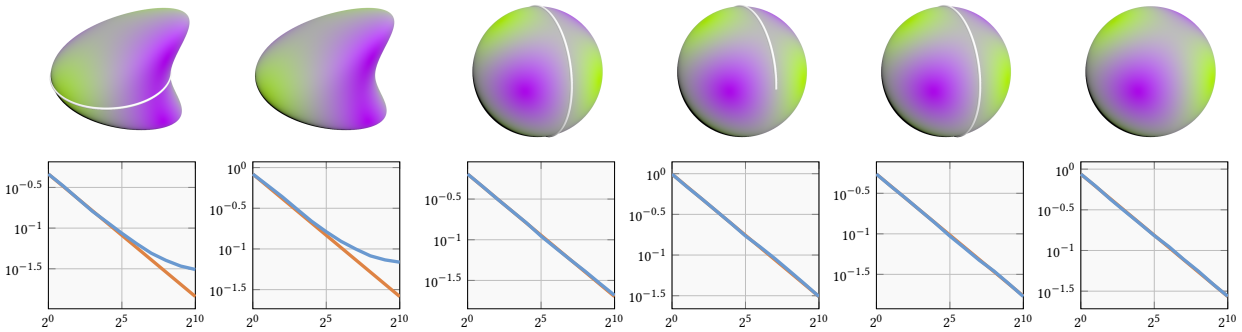
(c) Z-Order ($s = 1.15$) Ends Poisson

(d) Z-Order ($s = 1.05$) Ends Poisson

(e) Circle Two-sided Poisson

(f) Torus Torus Knot Laplace

Figure 6.4: Error convergence. This figure continues to Fig. 6.5. In each of the examples, we show the visualization of the scene setup (top), error convergence plot (middle), and scene description (bottom). The scene visualizations show the analytical or reference solution of the problem by mapping the range of solution values to the green-to-purple color gradient and placing a white point or curve on the Dirichlet boundary. The vertical axis of the error plot shows the root mean squared error of the estimates at a few points on the surface in a logarithmic scale, and the horizontal axis shows the number of samples N_P in a logarithmic scale. The blue curves show the result of the experiment, and the orange curves show a line that corresponds to the desired $\mathcal{O}(1/\sqrt{N_P})$ convergence rate. The scene description indicates the surface shape (top), boundary shape/type (middle), and problem type (bottom). While the expected $\mathcal{O}(1/\sqrt{N_P})$ is achieved in most scenes, the bias remains high when an aggressive medial axis pruning parameter is used (c) and when the source term of the problem is relatively complex (e, g, and h).



(g) Dziuk Surface Closed Screened Poisson

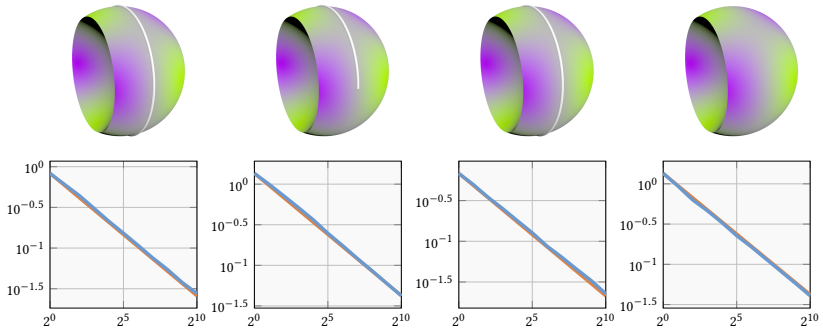
(h) Dziuk Surface No Boundary Screened Poisson

(i) Sphere Closed Poisson

(j) Sphere Open Poisson

(k) Sphere Closed Screened Poisson

(l) Sphere No Boundary Screened Poisson



(m) Punched Closed Poisson

(n) Punched Open Poisson

(o) Punched Closed Screened Poisson

(p) Punched No Boundary Screened Poisson

Figure 6.5: Fig. 6.4 continued.

$N_V = 0$. While the method shows the expected convergence rate of $\mathcal{O}(1/\sqrt{N_P})$ in most examples, we observed a relatively large bias with problems having more complex source term functions, indicating the need for future work to investigate estimating $g(\mathbf{x})$ from Eq. 6.4 for such problems.

We used the following problems to generate the error convergence plots in Fig. 6.4. Note that we finely discretized the surfaces we describe below to obtain the data we show in the figure.

(a) The helix curve we use has three turns, has a radius of 1, and the endpoints have a height difference of 2. We solve the Laplace equation defined along the curve length ϕ as

$$\begin{aligned} \frac{\partial^2 u_{\mathcal{S}}}{\partial \phi^2} &= 0, \\ u_{\mathcal{S}}(0) &= 0, \\ u_{\mathcal{S}}(\psi) &= 1, \end{aligned} \tag{6.9}$$

where the boundary conditions are specified at the two ends of the curve, $\phi = 0$ and $\phi = \psi$. The analytical solution is $u_{\mathcal{S}}(\phi) = \phi/\psi$.

(b) to (d) The problem we solve is defined along the curve length ϕ as

$$\begin{aligned} \frac{\partial^2 u_{\mathcal{S}}}{\partial \phi^2} &= 0.02, \\ u_{\mathcal{S}}(0) &= 0, \\ u_{\mathcal{S}}(\psi) &= 1, \end{aligned} \tag{6.10}$$

where the boundary conditions are specified at the two ends of the curve, $\phi = 0$ and $\phi = \psi$, similar to (a). The analytical solution is $u_{\mathcal{S}}(\phi) = 0.01\phi^2 + \frac{1-0.01\psi^2}{\psi}\phi$. The helix curve in (b) is identical to the one in (a). The z-order curve in (c) and (d) is defined using 8 points, $(\pm 1.0, \pm 1.0, \pm 1.0)$.

(e) This scene is one of the scenes in the grid-based CPM paper by King et al. [2024]. On a unit circle, we have a two-sided Dirichlet boundary. In polar coordinates, the problem we solve in terms of the angle θ is

$$\begin{aligned} \frac{\partial^2 u_{\mathcal{S}}}{\partial \theta^2} &= -2\cos(\theta - \theta_c), \\ u_{\mathcal{S}}(\theta_c^-) &= 2, \\ u_{\mathcal{S}}(\theta_c^+) &= 22, \end{aligned} \tag{6.11}$$

where $\theta_c = 1.022\pi$ is the position of the Dirichlet boundary. The analytical solution to this problem is $u_{\mathcal{S}}(\theta) = 2\cos(\theta - \theta_c) + \frac{10}{\pi}(\theta - \theta_c)$.

(f) The surface we used is a torus with a major radius $R = 3$ and a minor radius $r = 1$. The Dirichlet boundary curve is a torus knot expressed as a parametric curve

$$x_1(s) = v(s)\cos(as), \quad x_2(s) = v(s)\sin(as), \quad x_3(s) = \sin(bs), \tag{6.12}$$

where $v(s) = R + \cos(bs), a = 3, b = 7$, and $s \in [0, 2\pi]$. We solve the Laplace equation on the torus with boundary condition $\sin(s)$ along the curve. We used the grid-based CPM implementation of King et al. [2024] with a grid spacing of 0.02 to generate a reference solution and measured the error of PWoS against it.

(g) and (h) The surface we used for these setups is the one given by Dziuk [1988] and later used in multiple CPM works [Chen and Macdonald 2015; King et al. 2024]. The surface is expressed as $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^3 | (x_1 - x_3)^2 + x_2^2 + x_3^2 = 1\}$. The problem we solve is

$$\nabla^2_{\mathcal{S}} u_{\mathcal{S}}(\mathbf{x}) - u_{\mathcal{S}}(\mathbf{x}) = -f_{\mathcal{S}}(\mathbf{x}), \quad \mathbf{x} \in \mathcal{S}, \tag{6.13}$$

where $f_{\mathcal{S}}$ has an analytical, yet complex, expression we can derive as in the work by Chen and Macdonald [2015], so the solution of the problem becomes $u_{\mathcal{S}}(\mathbf{x}) = x_1 x_2$. For (g), we use a unit circle on the x_1x_2-plane with the analytical solution specified on it as the boundary value as the Dirichlet boundary. For (h), we did not use any boundary to show the algorithm's convergence for the screened Poisson equation without any boundaries.

(i) to (p) The scenes consider the unit sphere with a spherical harmonic function as the analytical solution, as is done in the study of mesh Laplacians [Bunge and Botsch 2023]. The sphere mesh is punched inward at $x_3 = 0.25$ for (m) to (p) to test the algorithm on a geometry with sharp corners. Given a spherical harmonic $Y_2^3(\mathbf{x}) = \frac{1}{4}\sqrt{\frac{105}{\pi}}(x_1^2 - x_2^2)^2 x_3$ with eigenvalue -12 as the solution, we solve the Poisson equation

$$\nabla^2_{\mathcal{S}} u_{\mathcal{S}}(\mathbf{x}) = -12 Y_2^3(\mathbf{x}), \quad \mathbf{x} \in \mathcal{S}, \tag{6.14}$$

for (i), (j), (m), and (n) and the screened Poisson equation

$$\nabla^2_{\mathcal{S}} u_{\mathcal{S}}(\mathbf{x}) - u_{\mathcal{S}}(\mathbf{x}) = -13 Y_2^3(\mathbf{x}), \quad \mathbf{x} \in \mathcal{S}, \tag{6.15}$$

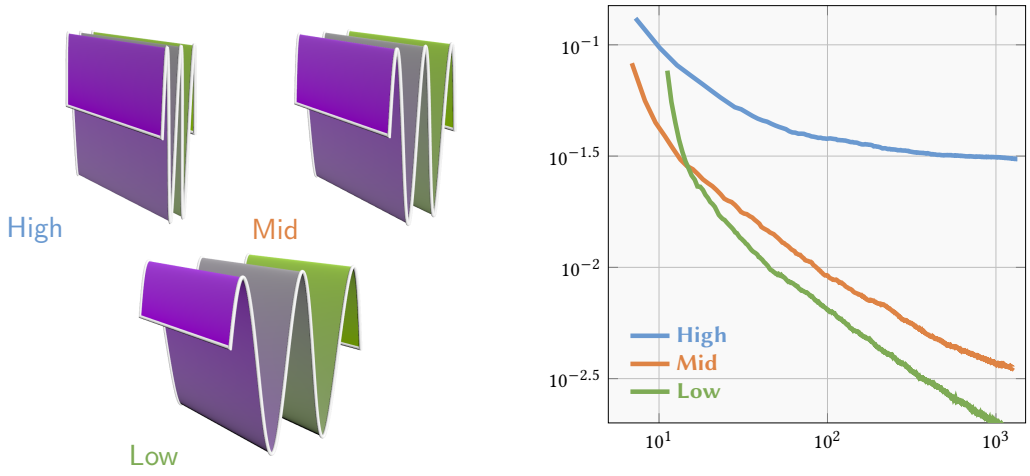


Figure 6.6: Effect of local feature size on convergence speed. We bend a rectangular strip of size 10 by 2 units along sinusoidal curves with high, middle, and low frequencies (top left, top right, and bottom, respectively, in the visualization) and solve the Laplace equation. The analytical solution is defined as the distance along the longer edge of the strip from one of the shorter edges. The vertical axis of the convergence plot represents the root mean squared error (RMSE), while the horizontal axis shows the time in seconds measured on a MacBook Pro with an M1 Pro chip. We used 1000 sample evaluation points. The geometry with a larger local feature size allows for faster convergence with lower bias.

for (k), (l), (o), and (p). For (i), (k), (m), and (o), we use the unit circle on the x_1x_2-plane as the Dirichlet boundary, and for (j) and (n), we use the unit semicircle where $x_2 > 0$ as the Dirichlet boundary. We observe the expected convergence behavior with all of the cases in (i) to (p) and suspect that it has something to do with the fact that the source term is a constant multiple of the solution.

Local Feature Size and Error

Fig. 6.6 compares the convergence of our method for problems with different local feature sizes. We observe that a larger local feature size corresponds to faster convergence with lower bias.

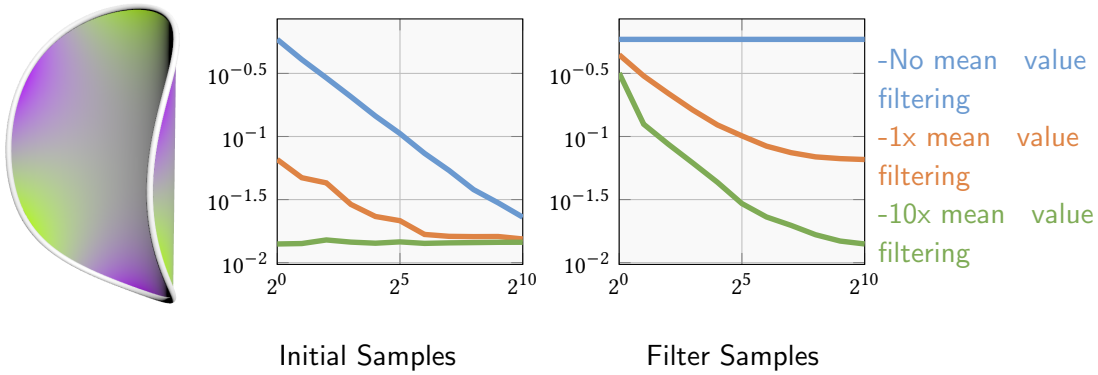


Figure 6.7: Mean value filtering error. For the Laplace equation with solution $u = r^3 \sin(3\theta)$ in polar coordinates defined on a unit disk, we curve the disk and measure the error of PWoS with mean value filtering. In each of the two plots, we compare the error without mean value filtering against one application of filtering and ten applications of filtering. The left plot shows the root mean squared error when changing the number of sample paths used to generate the initial estimate with a fixed number of filter samples (1024). The right plot shows the root mean squared error when changing the number of samples used to construct the filter and fixing the number of sample paths used to generate the initial estimate to 1. We observe that applying the mean value filter constructed with a sufficiently large number of samples can reduce the error significantly, even when the initial estimate is constructed with a small sample path count.

Mean Value Filtering

We run the mean value filtering algorithm on a Laplace equation on a triangulated curved disk surface in Fig. 6.7. In this setup, we observe that applying the filter multiple times with a filter constructed with a sufficiently high sample count can significantly reduce the error, even when the initial estimate is computed with a small number of samples. As expected, the filter is more effective when constructed with more samples, but the error decreases slower than the rate $\mathcal{O}(1/\sqrt{N_P})$, where N_P is the number of samples used to construct the filter. As no recursive estimation is required with the mean value filtering, it significantly improves the efficiency of PWoS.

6.4.2 Applications

Diffusion Curves

Diffusion curves [Orzan et al. 2008] succinctly represent an image as a collection of curves with associated colors. The final image, exhibiting smooth color gradients, is recovered by solving a Laplace equation with the curves dictating boundary conditions. In our application, we solve the surface Laplace equation using PWoS. With our approach, the surface need not have a boundary curve conforming to the discretized mesh, which contrasts with the common approach [De Goes et al. 2022]. Fig. 6.8 shows the reconstruction of color at each point on the discretized surface, represented as a quadrilateral mesh and a combination of triangles, polylines, and point clouds. Our method naturally supports two-sided boundaries, with different colors specified on each side of a curve, and surface geometries with mixed codimension. Additionally, the pointwise nature of PWoS allows it to be applied in a view-dependent manner. For example, given a camera configuration, for antialiasing, we sample points within each pixel to generate rays. We then generate PWoS samples at the ray-surface intersection points. No computational resources are wasted on surface points that are invisible to the camera (Fig. 6.1), and we can obtain clean results without relying on a fine discretization of the surface. Boundary integral-based approaches [Bang et al. 2023; Sun et al. 2012] would similarly allow domain-discretization-free evaluation of diffusion curves, but they first require a global linear system solve. Moreover, such methods are not applicable to general curved surfaces, and would need to map the results in the 2D domain to the surface via UV coordinates, for example.

Geodesic Distance

Crane et al. [2013] proposed the heat method, which solves two standard surface PDEs in series to compute the geodesic distance from the boundary $\mathcal{C}$. The steps are summarized as

1. $\nabla^2_{\mathcal{S}} u_{\mathcal{S}} - (1/t) u_{\mathcal{S}} = 0$ where $u_{\mathcal{S}} = 1$ on $\mathcal{C}$,
2. $\mathbf{X} = -(\nabla_{\mathcal{S}} u_{\mathcal{S}}) / \|\nabla_{\mathcal{S}} u_{\mathcal{S}}\|_2$, and
3. $\nabla^2_{\mathcal{S}} \phi_{\mathcal{S}} = \nabla_{\mathcal{S}} \cdot \mathbf{X}$ where $\phi_{\mathcal{S}} = 0$ on $\mathcal{C}$,

where t is a small positive constant and ϕ is the geodesic distance. Step 2 uses the gradient of the solution to the screened Poisson equation found in Step 1. With a discretization-based method, a discrete gradient operator is used to estimate this gradient; in our method,

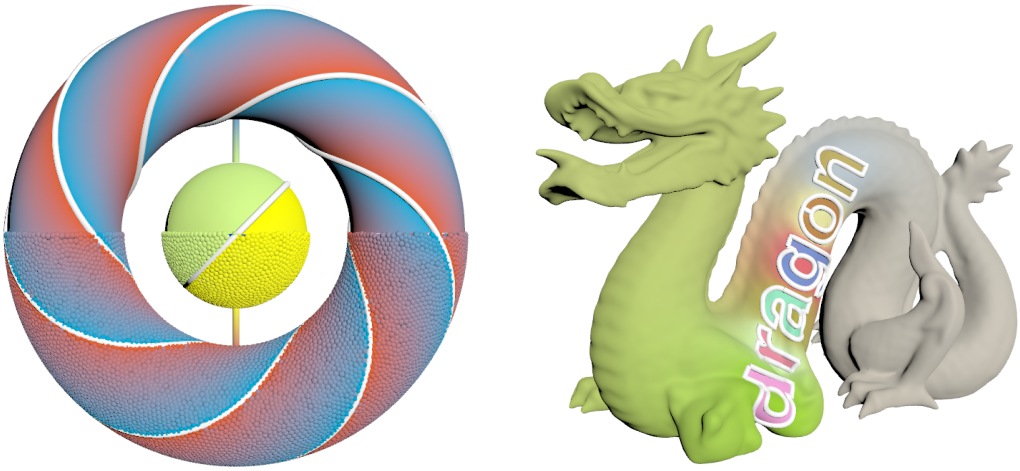


Figure 6.8: Surface diffusion curves solved on various surface representations. The surface on the left is represented as a combination of triangles, polylines, and oriented points; the surface on the right is represented as a quadrilateral mesh. The scene on the left, featuring surface geometry of mixed codimension, was adapted from the work of King et al. [2024].

we directly evaluate the gradient of u during Step 1 using the method in Section 6.2.3, without needing u itself. We evaluate the gradient at mesh vertices and normalize it to get $\mathbf{X}$ at mesh vertices. In Step 3, again, we do not rely on a discrete divergence operator to solve the Poisson equation, but instead use the method described in Section 6.2.3. When our Poisson solver requires the evaluation of $\mathbf{X}$ at a point, we interpolate $\mathbf{X}$ from the mesh vertices and (re)normalize it. We can similarly compute the geodesic distance on a surface represented as a point cloud. Fig. 6.9 compares our PWoS-based heat method on surfaces represented as polygonal meshes or oriented point clouds against the heat method with grid-based CPM [King et al. 2024] and the exact geodesic distance computed with Geometry Central [Sharp et al. 2019]. Our results are consistent with the reference implementation, albeit with minor deviations.

Surface Wave Animation

Using our screened Poisson equation solver, we can solve some classes of time-dependent problems. We discretize the wave equation in time with implicit Euler to get a screened Poisson equation and solve it with time stepping (Fig. 6.10). At each time step, we store the solution at the vertices of the mesh and query the solution from previous frames by interpolating the values. In contrast to grid-based CPM [Auer et al. 2012], our method

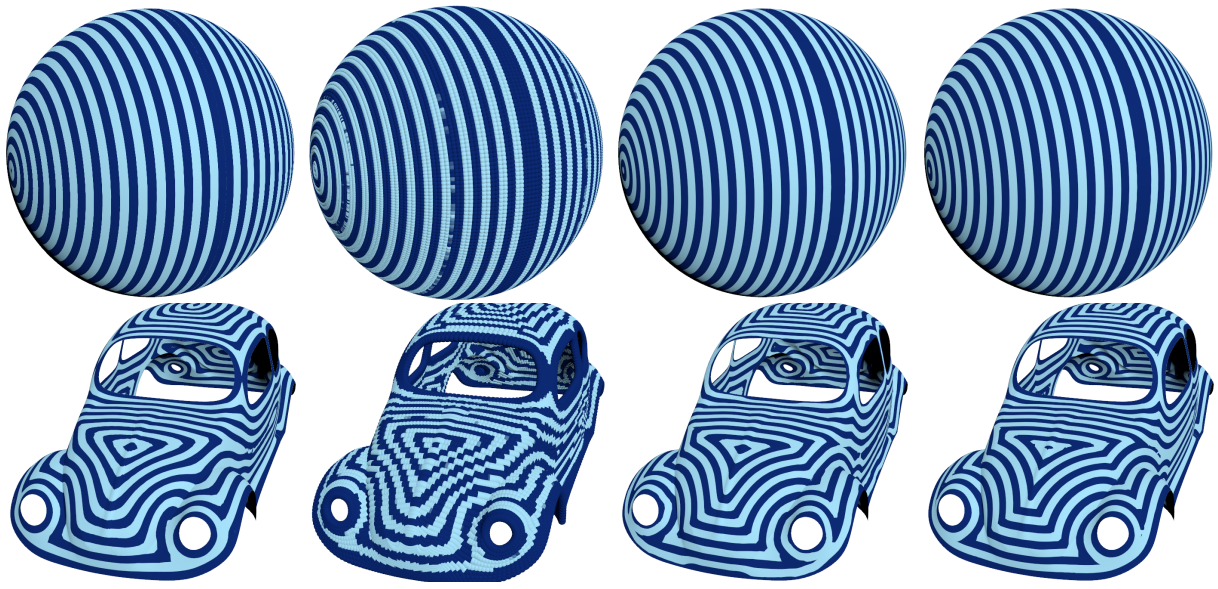
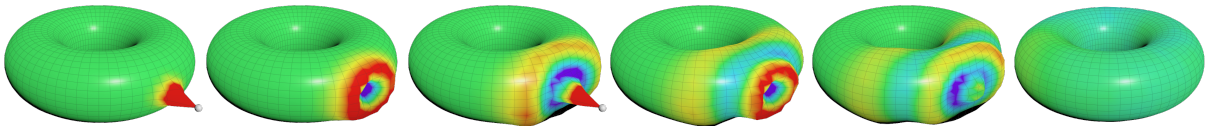


Figure 6.9: Geodesic distance computation with the heat method. For each of the two scenes, we compare our algorithm on a polygonal mesh representation (leftmost) and oriented point cloud representation (middle-left) against a grid-based CPM counterpart [King et al. 2024] (middle-right) and the exact polyhedral distance computed with Geometry Central [Sharp et al. 2019] (rightmost). For the sphere surface (top), we compute the distance from the circle boundary curve in the center, and for the car surface (bottom), we compute the distance from the surface boundary edges. Note that the rendering of the point clouds assigns a UV coordinate per point, resulting in larger visual differences.



Frame 7 Frame 19 Frame 31 Frame 43 Frame 55 Frame 200

Figure 6.10: Surface wave animation. We solve the wave equation on the surface and visualize the solution as a displacement applied on the surface in the normal directions. Combined with a time-stepping approach, our method can be applied to a few time-dependent problems. We set the solution to one point on the mesh as the boundary condition for the first 49 frames, and remove the boundary from the 50th frame. The waves damp out as the simulation continues, as expected.

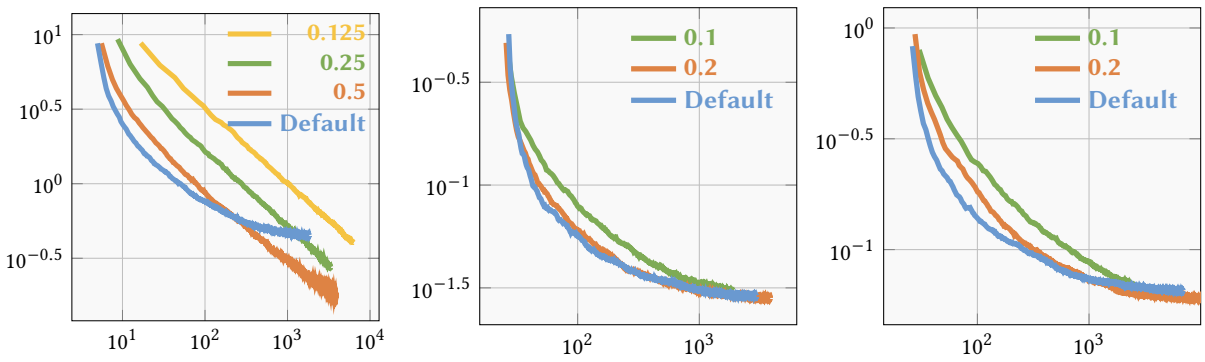
directly deals with surface geometry without defining an embedding grid.

6.4.3 Performance

The performance of our method depends on several factors; we report timings for two representative examples. We measured these timings using a workstation with two Intel(R) Xeon(R) Silver 4316 CPUs, each with 20 CPU cores. For the scene in Fig. 6.1 bottom left, the image resolution is 640 by 480 and the number of samples per pixel was 1024. The precomputation step, including medial axis computation, took less than 1 minute, and the rest of the main parts of PWoS took 2 hours and 11 minutes. We did not apply mean value filtering. For the scene in Fig. 6.8 left, we have 28,119 evaluation points. The medial axis point cloud extraction took 2.4 seconds, the initial solution estimate with 1 sample took 1.3 seconds, and the application of 10 mean value filtering steps with a filter constructed with 128 samples took 13 seconds. Optimizing the implementation with GPU acceleration may further improve performance.

6.5 Discussion

We have developed a Monte Carlo method for surface PDEs by augmenting the formulation of WoS with a closest point projection step. Our algorithm is justified through its connection to the theory of closest point extensions drawn from the CPM literature. To accelerate its convergence, we have developed a practical mean value filtering method that utilizes a discrete basis defined over the surface. We have further analyzed the method's



(e) (g) (h)

Figure 6.11: Using bounded sphere size. For the Poisson and screened Poisson problems (e), (g), and (h) in Fig. 6.4, we compare the Default option of not constraining the sphere size (apart from the limit imposed by the local feature size estimate) against specified limits on the maximum sphere size as indicated in the legend. The vertical axis shows the root mean squared error, and the horizontal axis shows the time in seconds measured on a MacBook Pro with an M1 Pro chip. For (e), we had 1024 evaluation points, and for (g) and (h), we used 100 sample evaluation points. While limiting the sphere size may reduce the bias, as we can observe from the intersections of the curves for the default option and the curves for the sphere-size-constrained option, the computation may take longer, and it is difficult to get a practical advantage.

convergence on representative analytical tests and demonstrated its application to graphics problems.

PWoS currently supports only Dirichlet boundary conditions; efficient Neumann or Robin boundary handling similar to Walk on Stars for volumetric PDEs [Sawhney and Miller et al. 2023; 2024b] would require the availability of a few more queries, such as a ray intersection query against the (extended) boundaries.

While we used a local feature estimation algorithm to allow walks with larger step sizes, the local feature size estimation itself imposes additional smoothness assumptions on the surface. To respect small-scale local features, the walk can require many iterations to reach a Dirichlet boundary. This effect is partly due to our algorithm (like WoS) being based on an integral equation that holds only locally inside a ball. Revisiting this choice using an integral equation based on a global relationship, such as the one underpinning WoB, could lead to a more efficient alternative for surface PDEs.

Lastly, our method relied on the assumption that the closest point extension compensation term (i.e., $g(\mathbf{x})$ in Eq. 6.4) in the embedding PDE is negligible. We empirically showed that the algorithm designed with this assumption works well when the source term has a relatively simple expression, but we do not yet have a complete understanding of when this assumption is strictly valid. However, since $g(\mathbf{x})$ tends to zero continuously as $\mathbf{x}$ approaches the surface, the influence of ignoring this term is expected to decrease as we shrink the embedding space (i.e., shrink the sphere size). One can always take a smaller sphere size, albeit at a higher computational cost, as we show in Fig. 6.11. Extending our method to consider the effect of the compensation term would further improve the reliability and broaden the applicability of our method.

Chapter 7

Conclusion and Future Work

7.1 Summary

We have developed spatial discretization-free, pointwise Monte Carlo solvers for partial differential equations (PDEs) that address limitations of prior methods.

The Walk on Boundary (WoB) method solves second-order linear PDEs under a variety of boundary conditions. We have demonstrated its applicability to elliptic, scalar-valued equations (Poisson), time-dependent parabolic equations (diffusion), and vector-valued equations (Stokes). Beyond introducing WoB to the graphics community, we highlighted its striking similarity in formulation to Monte Carlo rendering. This analogy enables the direct application of advanced variance reduction techniques and facilitates implementation on top of existing ray tracing codebases.

While much of the community's focus has been on extending Monte Carlo techniques to a broader class of linear PDEs, we have taken a step further by investigating their use for nonlinear PDEs, specifically the incompressible Navier–Stokes equations for fluid simulation. Using operator splitting, we decomposed these nonlinear equations into simpler substeps and designed pointwise solvers for each, leveraging both vorticity- and velocity-based formulations.

In addition, we developed the Projected Walk on Spheres (PWoS) method, which combines the Walk on Spheres (WoS) algorithm with the Closest Point Method. PWoS is the first Monte Carlo method capable of solving surface PDEs without requiring explicit surface parametrizations.

With these Monte Carlo techniques, for a wider variety of PDE problems, we can now obtain solutions at a few desired points without assembling or solving global linear systems. These methods are trivially parallelizable and rely only on simple geometric operations, such as closest point and ray intersection queries, without the need for conforming discretizations or cut-cell treatments of complex geometries. These properties make Monte Carlo PDE solvers a compelling alternative to traditional discretization-based approaches.

This thesis has taken important steps toward developing general Monte Carlo PDE solvers for graphics applications. However, it has only begun to explore the full potential of these methods. Below, we outline several promising directions for future research.

7.2 Applicability of Monte Carlo PDE Solvers

There has been an extensive study of Monte Carlo PDE solvers for second-order linear elliptic and parabolic PDEs, and we have discussed some methods in these categories, with an additional application of them to nonlinear Navier–Stokes equations. Still, we identify two classes of equations that remain difficult to deal with using Monte Carlo methods.

7.2.1 Hyperbolic Equations

The first category is (linear) hyperbolic equations, such as the scalar wave equation and elastodynamic wave equations. Mascagni [2011] notes that designing general-purpose Monte Carlo PDE solvers for hyperbolic equations is challenging, except in certain special cases like the telegrapher's equation and Burger's equation. This difficulty arises because Monte Carlo solvers for elliptic and parabolic equations are typically grounded in Brownian motion, which models the diffusion of quantities. In contrast, hyperbolic equations exhibit wave-like behavior, propagating sharp features without diffusion, making stochastic interpretations less natural. While one could discretize time and apply Monte Carlo solvers to each time slice, such finite-difference approaches may introduce undesirable bias into the estimates.

This limitation primarily applies to bottom-up approaches like WoS. In contrast, we believe that the WoB methodology has the potential to generalize to hyperbolic equations. Many hyperbolic PDEs admit formulations as time-domain boundary integral equations (BIEs), similar to the parabolic case—a fact well known in the boundary element method community [Dominguez 1993; Schanz 2001]. The key challenge then becomes devising appropriate sampling strategies for these time-domain BIEs. This is nontrivial due to the

appearance of Heaviside and Dirac delta functions in the kernels, but a principled Monte Carlo treatment still appears feasible.

An alternative approach is to transform the original time-domain problem into a collection of elliptic problems via the Laplace transform. Since both the forward and inverse Laplace transforms are expressed as integrals, this representation can open up another promising direction for designing pointwise time-domain Monte Carlo PDE solvers for hyperbolic equations.

7.2.2 General Nonlinear Equations

In our Navier–Stokes solver, we follow a common operator splitting approach of handling nonlinear advection explicitly as a separate step. A similar idea appears in other domains—for example, in nonlinear deformable body simulation, where some prior work has proposed linearizing the governing equations and advancing the system using small timesteps [Macklin et al. 2019; Müller and Gross 2004; Müller et al. 2002]. Further study is needed to evaluate the validity of such linearizations under different conditions and to quantify their effects on solution accuracy and convergence.

Beyond these specific scenarios, general-purpose Monte Carlo methods for directly solving nonlinear equations would be valuable in a variety of applications. For example, anisotropic diffusion in image filtering involves diffusion coefficients that depend nonlinearly on the solution [Catté et al. 1992; Perona and Malik 1990].

In many cases, nonlinearities can be absorbed into linear boundary integral formulations as a solution-dependent source term, as demonstrated in the nonlinear Poisson equation [Sabelfeld and Simonov 1994, Chapter 7] and in geometrically nonlinear elasticity simulations within the BEM framework [Deng et al. 2018]. Still, nonlinear equations present a significant challenge for Monte Carlo PDE solvers, particularly in terms of efficiency. When nonlinearities appear as products of functions of the unknown solution, unbiased Monte Carlo estimates require independently sampled evaluations of each factor. This independence is necessary to avoid bias but substantially increases the computational cost. Although such an approach has been partly explored by Bossy et al. [2015], it can easily become costly.

As noted by Sabelfeld and Simonov [1994, Chapter 7], another strategy is to estimate the solution of the linearized PDE at a number of sample points using a Monte Carlo solver, then refine these estimates via fixed-point iteration. However, this iterative process may converge slowly, and analyzing its error behavior remains difficult.

Within computer graphics, stylized rendering problems often involve similar types of nonlinearities [Tong and Hachisuka 2025; West and Mukherjee 2024], and insights from these methods could also inform the development of nonlinear Monte Carlo solvers.

7.3 Improvement of Core Monte Carlo PDE Solver

In addition to broadening the range of applications for Monte Carlo PDE solvers, we believe there is still considerable room for improving the efficiency of core solvers—even for simple cases such as the Laplace or Poisson equations—beyond what we have discussed and beyond the concurrently proposed WoS(t) methods. While the concurrent works presented in Section 2.3.5 primarily focus on improving efficiency by better utilizing or combining pointwise estimates of the solution given by an underlying Monte Carlo PDE solver, we argue that this underlying solver itself can be further enhanced. In contrast to WoS(t) where the choice of integral equations and the sampling strategies are tighly coupled with less room for improvement, WoB is a general design framework that leaves a lot of room for exploration. In the following, we discuss concrete directions for improving the WoB method by addressing its current key limitations.

7.3.1 Choice of Boundary Integral Equations

We have discussed how different boundary integral equations (BIEs) can be leveraged for different boundary condition types. For pure Dirichlet or pure Neumann problems, the formulations we presented yield well-behaved algorithms in the sense that the path throughput (i.e., the multiplicative weight assigned to each sampled path) remains bounded when using ray intersection sampling to trace all boundary interactions by splitting. However, this favorable behavior breaks down for mixed or Robin boundary conditions, and the practicality of our method becomes limited.

When we turn our attention to the boundary element method (BEM), a discretization-based counterpart to WoB, it combines the conventional direct BIE with another BIE called the hypersingular BIE to construct algorithms with symmetric system matrices [Liu et al. 2012]. While symmetry is not a primary concern for WoB, the idea of combining both BIEs is potentially valuable in our context, especially for controlling path throughput. For example, in mixed boundary problems, we could apply the conventional BIE on Neumann boundaries and the hypersingular BIE on Dirichlet boundaries. This combined formulation enables a more structured analysis of the behavior of the integral kernels involved, which may guide the design of a more stable WoB variant.

In particular, the Calderón operator [Pechstein 2013], which unifies these two BIEs, is known to act as a projection operator. This property may prove useful in constructing formulations with inherently bounded throughput, thereby improving the practicality of WoB for complex boundary conditions.

7.3.2 Sampling

The sampling techniques discussed in Chapter 3 represent only a small subset of many strategies that could be explored. In particular, ray intersection sampling with stochastic selection of a single intersection point introduces an exponential increase in variance with respect to the recursion depth of the WoB algorithm, even for simple cases such as the Laplace equation with only Dirichlet boundaries. In such scenarios, the current WoB implementation can easily become less efficient than WoS(t) (Fig. 3.8). An alternative is to take all intersection points by splitting paths, but doing this naïvely results in an exponential number of split paths. While we could explore non-ray-based sampling strategies, for pure Dirichlet or Neumann problems, ray-intersection-based sampling is attractive because dividing by the sampling probability density function (PDF) cancels the singularity in the kernel function—a desirable property we would like to retain.

One potentially effective sampling strategy is to always consider all intersection points along each ray, while avoiding full path branching. At each step, we collect all N intersection points along the current ray. Then, we randomly select one of these N points and cast a new ray from it, yielding another M intersection points. We then evaluate all $N \times M$ pairwise combinations between the current and next sets of points to estimate path contributions. These combinations can be weighted using multiple importance sampling (MIS). Specifically, we compute the probability of obtaining the sampled M points not only from the chosen point, but also by marginalizing over the probabilities of selecting each of the N candidate points that could have been used to cast the ray. This strategy avoids exponential growth in the number of path branches while still accounting for all interactions along each ray. Furthermore, since the sampled points at each level lie on a line, we expect the kernel singularities to cancel due to the structure of the sampling PDF, preserving a key advantage of ray-intersection-based techniques.

For mixed or Robin boundary conditions, especially when employing a combined formulation using both conventional and hypersingular BIEs as in Section 7.3.1, we may also need to generate both forward and backward paths and merge them. In such cases, techniques from bidirectional Monte Carlo rendering, such as UPS/VCM [Georgiev et al. 2012; Hachisuka et al. 2012], could provide a promising foundation for designing efficient algorithms.

7.3.3 Series Truncation and Path Weights

In Section 3.2.1, we discussed a path truncation strategy that assigns a weight of one-half to the longest path, based on structural properties of interior Dirichlet problems. We also mentioned that more sophisticated approaches, such as series acceleration techniques, could be used to assign more elaborate weights to paths of different depths in order to improve efficiency. In this thesis, however, we employed only the simple truncation strategy across all problem types.

Designing such depth-dependent weights is a nontrivial problem. A formal treatment requires analyzing the spectral properties of the associated integral operator, in a manner analogous to eigenvalue analysis for matrices. These spectral characteristics are highly problem-dependent and directly influence the convergence behavior of the series expansion. This analysis becomes particularly important for equations like the Helmholtz equation, where carefully assigning path weights is crucial for constructing a valid estimator. Similar challenges are expected when dealing with mixed or Robin boundary conditions using the combined BIE approach discussed in Section 7.3.1. However, estimating the eigenvalues of the integral operator is generally difficult and often requires either analytical derivation or dedicated numerical techniques [Sabelfeld 1991, Section 3.1.4].

Thus, a unified, semi-automatic approach to assigning weights to sample path contributions would be desirable. One idea is to determine these weights adaptively based on the sampled path contributions themselves, rather than predetermining them. In the context of the boundary element method (BEM), a related concept appears in the form of the generalized minimal residual method (GMRES) [Saad and Schultz 1986]. While the implementation details of GMRES differ, its theoretical foundation is to find a solution expressed as a linear combination of basis vectors spanning a Krylov subspace. Translating this idea to our setting, suppose we can separately estimate contributions from paths of different lengths for a given integral equation; i.e., using the notation in Section 3.2.1, we have estimates of $H^i\overline{u}_D$ for $i = 0, \ldots, M$. We then seek a linear combination of these contributions that minimizes the residual in the least squares sense: $\operatorname{argmin}_{w_i} \left\| \overline{u}_D - (I - H) \sum_{i=0}^{M} w_i H^i \overline{u}_D \right\|_2^2$. Implementing this approach in the Monte Carlo framework would require considering how to efficiently evaluate additionally necessary quantities, such as inner products between contributions from paths of different lengths. This direction presents an interesting possibility for automatically assigning weights to paths of varying lengths, potentially improving estimator efficiency and convergence for various types of problems.

7.4 Utilization of Discretization-Based Methods

Even though our Monte Carlo PDE solvers offer unique advantages, such as pointwise evaluation of the solution without requiring a global linear system solve, we do not expect them to replace existing spatial discretization-based solvers, such as boundary element, finite difference, finite element, or finite volume methods, in all cases. These traditional approaches remain particularly effective when a dense evaluation of the solution across the entire domain is desired. In fact, many of these methods benefit from advanced techniques such as adaptive hierarchical data structures that significantly improve their efficiency. As such, selecting the most suitable method often depends on the specific requirements of the application.

Importantly, the decision between Monte Carlo methods and discretization-based solvers is not binary.

Monte Carlo techniques can often be integrated into traditional methods. For example, Gipson [1985] employed Monte Carlo integration to evaluate the volume integrals arising from source terms in BEM. Madan et al. [2025] proposed a Monte Carlo traversal of tree structures to accelerate treecode algorithms for fast approximate dense matrix multiplications. Sabelfeld and Agarkov [2024, 2025] considered stochastic evaluations of matrix entries in BEM-type framework.

Conversely, discretely stored information can be used to improve the efficiency of Monte Carlo solvers. Sabelfeld [2012], for instance, proposed using a solution obtained by the method of fundamental solutions (MFS) as a control variate within WoB. Since the MFS yields an approximate solution that satisfies the governing PDE exactly but may violate boundary conditions, WoB can be used to correct the residual boundary errors. More recently, Shalimova and Sabelfeld [2025] iteratively refined the boundary density functions estimated at discrete points, using the current density estimate as a control variate. An analogous strategy for solutions from domain-discretization-based solvers (e.g., finite difference, finite volume, finite element) is less straightforward, as such solvers produce solutions that satisfy the PDE only in a discrete sense rather than continuously. However, if we can interpret these numerical results as a continuous solution perturbed by some error from the desired solution, they too would be exploitable as control variates.

These examples demonstrate that there remains significant room for exploration in hybrid approaches that combine stochastic and deterministic techniques. Identifying the optimal balance between them for a given problem will be key to realizing their full potential.

References

Y. Abousleiman and A.H.-D. Cheng. Boundary element solution for steady and unsteady stokes flow. *Computer Methods in Applied Mechanics and Engineering*, 117(1):1–13, 1994. doi:10.1016/0045-7825(94)90074-4.

Carlos J.S. Alves, Rodrigo G. Serrão, and Ana L. Silvestre. Fundamental solutions for the stokes equations: Numerical applications for 2d and 3d flows. *Applied Numerical Mathematics*, 170:55–73, 2021. doi:10.1016/j.apnum.2021.07.011.

N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999. doi:10.1007/PL00009475.

Ryoichi Ando, Nils Thuerey, and Chris Wojtan. A stream function solver for liquid simulations. *ACM Trans. Graph.*, 34(4), July 2015. doi:10.1145/2766935.

Alexis Angelidis and Fabrice Neyret. Simulation of smoke based on vortex filament primitives. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 87–96, 2005. doi:10.1145/1073368.1073380.

James Arvo and David Kirk. Particle transport and image synthesis. *SIGGRAPH Comput. Graph.*, 24(4):63–66, September 1990. doi:10.1145/97880.97886.

S. Auer, C. B. Macdonald, M. Treib, J. Schneider, and R. Westermann. Real-time fluid effects on surfaces using the closest point method. *Computer Graphics Forum*, 31(6): 1909–1923, 2012. doi:10.1111/j.1467-8659.2012.03071.x.

Vinicius C. Azevedo, Christopher Batty, and Manuel M. Oliveira. Preserving geometry and topology for fluid flows with thin obstacles and narrow gaps. *ACM Trans. Graph.*, 35(4), July 2016. doi:10.1145/2897824.2925919.

Ghada Bakbouk and Pieter Peers. Mean Value Caching for Walk on Spheres. In *Eurographics Symposium on Rendering*, 2023. doi:10.2312/sr.20231120.

Seungbae Bang, Kirill Serkh, Oded Stein, and Alec Jacobson. An adaptive fast-multipole-accelerated hybrid boundary integral equation method for accurate diffusion curves. *ACM Trans. Graph.*, 42(6), December 2023. doi:10.1145/3618374.

Alec Bartsch, Colin Thompson, and Fernando de Goes. A procedural approach for stylized bark shading. In *ACM SIGGRAPH 2023 Talks*, 2023. doi:10.1145/3587421.3595419.

Mégane Bati, Stéphane Blanco, Christophe Coustet, Vincent Eymet, Vincent Forest, Richard Fournier, Jacques Gautrais, Nicolas Mellado, Mathias Paulin, and Benjamin Piaud. Coupling conduction, convection and radiative transfer in a single path-space: Application to infrared rendering. *ACM Trans. Graph.*, 42(4), July 2023. doi:10.1145/3592121.

Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph.*, 26(3):100–es, July 2007. doi:10.1145/1276377.1276502.

Yash Belhe, Michaël Gharbi, Matthew Fisher, Iliyan Georgiev, Ravi Ramamoorthi, and Tzu-Mao Li. Discontinuity-aware 2d neural fields. *ACM Trans. Graph.*, 42(6), December 2023. doi:10.1145/3618379.

Mikhail Belkin, Jian Sun, and Yusu Wang. Constructing laplace operator from point clouds in $\mathbb{R}^d$. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, page 1031–1040. Society for Industrial and Applied Mathematics, 2009. doi:10.1137/1.9781611973068.112.

Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Trans. Graph.*, 39(4), July 2020. doi:10.1145/3386569.3392481.

Blender Online Community. Blender 4.2, 2024. URL `https://www.blender.org/`. Computer Software.

Mireille Bossy, Nicolas Champagnat, Hélène Leman, Sylvain Maire, Laurent Violeau, and Mariette Yvinec. Monte carlo methods for linear and non-linear poisson-boltzmann equation. *ESAIM: Proc.*, 48:420–446, 2015. doi:10.1051/proc/201448020.

John C. Bowers, Jonathan Leahey, and Rui Wang. A ray tracing approach to diffusion curves. *Comput. Graph. Forum*, 30(4):1345–1352, 2011. doi:10.1111/j.1467-8659.2011.01994.x.

J.U. Brackbill and H.M. Ruppel. Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics*, 65(2): 314–343, 1986. doi:10.1016/0021-9991(86)90211-1.

Robert Bridson. *Fluid Simulation for Computer Graphics, Second Edition*. Taylor & Francis, 2015.

Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM Trans. Graph.*, 26(3):46–es, July 2007. doi:10.1145/1276377.1276435.

Tyson Brochu, Todd Keeler, and Robert Bridson. Linear-time smoke animation with vortex sheet meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 87–95, 2012. doi:10.2312/SCA/SCA12/087-095.

A. Bunge and M. Botsch. A survey on discrete laplacians for general polygonal meshes. *Computer Graphics Forum*, 42(2):521–544, 2023. doi:10.1111/cgf.14777.

Barbara Busnello, Franco Flandoli, and Marco Romito. A probabilistic representation for the vorticity of a three-dimensional viscous fluid and for general systems of parabolic equations. *Proceedings of The Edinburgh Mathematical Society*, 48(2):295–336, 2005. doi:10.1017/S0013091503000506.

Anne Campion-Renson and Marcel J. Crochet. On the stream function-vorticity finite element solutions of navier-stokes equations. *International Journal for Numerical Methods in Engineering*, 12(12):1809–1818, 1978. doi:10.1002/nme.1620121204.

Francine Catté, Pierre-Louis Lions, Jean-Michel Morel, and Tomeu Coll. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM Journal on Numerical Analysis*, 29(1):182–193, 1992. doi:10.1137/0729012.

Jeffrey N. Chadwick, Steven S. An, and Doug L. James. Harmonic shells: a practical nonlinear sound model for near-rigid thin shells. *ACM Trans. Graph.*, 28(5):1–10, December 2009. doi:10.1145/1618452.1618465.

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph.*, 36(4), July 2017. doi:10.1145/3072959.3073601.

Wesley Chang, Venkataram Sivaram, Derek Nowrouzezahrai, Toshiya Hachisuka, Ravi Ramamoorthi, and Tzu-Mao Li. Parameter-space restir for differentiable and inverse

rendering. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, 2023. doi:10.1145/3588432.3591512.

Jiong Chen, Fernando De Goes, and Mathieu Desbrun. Somigliana coordinates: an elasticity-derived approach for cage deformation. In *ACM SIGGRAPH 2023 Conference Proceedings*, 2023. doi:10.1145/3588432.3591519.

Jiong Chen, Florian Schäfer, and Mathieu Desbrun. Lightning-fast method of fundamental solutions. *ACM Trans. Graph.*, 43(4), July 2024. doi:10.1145/3658199.

Yujia Chen and Colin B. Macdonald. The closest point method and multigrid solvers for elliptic equations on surfaces. *SIAM Journal on Scientific Computing*, 37(1):A134–A155, 2015. doi:10.1137/130929497.

Alexander H.D. Cheng and Yongxing Hong. An overview of the method of fundamental solutions—solvability, uniqueness, convergence, and stability. *Engineering Analysis with Boundary Elements*, 120:118–152, 2020. doi:10.1016/j.enganabound.2020.08.013.

Albert Chern, Felix Knöppel, Ulrich Pinkall, Peter Schröder, and Steffen Weißmann. Schrödinger's smoke. *ACM Trans. Graph.*, 35(4), July 2016. doi:10.1145/2897824.2925868.

Ka Chun Cheung, Leevan Ling, and Steven J. Ruuth. A localized meshless method for diffusion on folded surfaces. *Journal of Computational Physics*, 297:194–206, 2015. doi:10.1016/j.jcp.2015.05.021.

Alexandre Joel Chorin. Numerical study of slightly viscous flow. *Journal of Fluid Mechanics*, 57(4):785–796, 1973. doi:10.1017/S0022112073002016.

Per H. Christensen. Adjoints and importance in rendering: An overview. *IEEE Trans. Vis. Comput. Graph.*, 9(3):329–340, June 2003. doi:10.1109/TVCG.2003.1207441.

Pascal Clausen, Martin Wicke, Jonathan R. Shewchuk, and James F. O'Brien. Simulating liquids and solid-liquid interactions with lagrangian meshes. *ACM Trans. Graph.*, 32(2), April 2013. doi:10.1145/2451236.2451243.

David L. Clements. A fundamental solution for linear second-order elliptic systems with variable coefficients. *Journal of Engineering Mathematics*, 49(3):209–216, July 2004. doi:10.1023/B:ENGI.0000031193.39633.d8.

Michael F. Cohen and Donald P. Greenberg. The hemi-cube: A radiosity solution for complex environments. *SIGGRAPH Comput. Graph.*, 19(3):31–40, July 1985. doi:10.1145/325165.325171.

Peter Constantin and Gautam Iyer. A stochastic lagrangian representation of the three-dimensional incompressible navier-stokes equations. *Communications on Pure and Applied Mathematics*, 61(3):330–345, 2008. doi:10.1002/cpa.20192.

Robert L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1): 51–72, January 1986. doi:10.1145/7529.8927.

Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, page 137–145, 1984. doi:10.1145/800031.808590.

Ricardo Cortez. The method of regularized stokeslets. *SIAM Journal on Scientific Computing*, 23(4):1204–1225, 2001. doi:10.1137/S106482750038146X.

Ricardo Cortez, Lisa Fauci, and Alexei Medovikov. The method of regularized Stokeslets in three dimensions: Analysis, validation, and application to helical swimming. *Physics of Fluids*, 17(3), 02 2005. doi:10.1063/1.1830486.

Georges-Henri Cottet and Petros D. Koumoutsakos. *Vortex Methods: Theory and Practice*. Cambridge University Press, Cambridge, 2000. doi:10.1017/CBO9780511526442.

Benoit Couët, Oscar Buneman, and Anthony Leonard. Simulation of three-dimensional incompressible flows with a vortex-in-cell method. *Journal of Computational Physics*, 39(2):305–328, 1981. doi:10.1016/0021-9991(81)90154-6.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph.*, 32(5), October 2013. doi:10.1145/2516971.2516977.

Ana Bela Cruzeiro. Stochastic approaches to deterministic fluid dynamics: A selective review. *Water*, 12(3), 2020. doi:10.3390/w12030864.

Michael Czekanski, Benjamin Faber, Margaret Fairborn, Adelle Wright, and David Bindel. Walking on spheres and talking to neighbors: Variance reduction for laplace's equation, 2025. URL https://arxiv.org/abs/2404.17692.

Fang Da, David Hahn, Christopher Batty, Chris Wojtan, and Eitan Grinspun. Surface-only liquids. *ACM Trans. Graph.*, 35(4), July 2016. doi:10.1145/2897824.2925899.

Ryan C. Daileda. The two dimensional heat equation. http://ramanujan.math.trinity.edu/rdaileda/teach/s12/m3357/lectures/lecture_3_6_short.pdf, 2012. Lecture Note.

Fernando de Goes and Mathieu Desbrun. Stochastic computation of barycentric coordinates. *ACM Trans. Graph.*, 43(4), July 2024. doi:10.1145/3658131.

Fernando De Goes, William Sheffler, and Kurt Fleischer. Character articulation through profile curves. *ACM Trans. Graph.*, 41(4), July 2022. doi:10.1145/3528223.3530060.

Auguste De Lambilly, Gabriel Benedetti, Nour Rizk, Chen Hanqi, Siyuan Huang, Junnan Qiu, David Louapre, Raphael Granier De Cassagnac, and Damien Rohmer. Heat simulation on meshless crafted-made shapes. In *Proceedings of the 16th ACM SIGGRAPH Conference on Motion, Interaction and Games*, 2023. doi:10.1145/3623264.3624457.

J.M Delaurentis and L.A Romero. A monte carlo method for poisson's equation. *Journal of Computational Physics*, 90(1):123–140, 1990. doi:10.1016/0021-9991(90)90199-B.

Freddy Delbaen, Jinniao Qiu, and Shanjian Tang. Forward–backward stochastic differential systems associated to navier–stokes equations in the whole space. *Stochastic Processes and their Applications*, 125(7):2516–2561, 2015. doi:10.1016/j.spa.2015.02.014.

Yani Deng, Junjie Rong, Wenjing Ye, and Lesley J Gray. An efficient grid-based direct-volume integration bem for 3d geometrically nonlinear elasticity. *Computational Mechanics*, 62(4):603–616, 2018. doi:10.1007/s00466-017-1515-z.

Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation '96*, pages 61–76, 1996. doi:10.1007/978-3-7091-7486-9_5.

Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, page 317–324, 1999. doi:10.1145/311535.311576.

Xinwen Ding and Adam R Stinchcombe. Walk-on-interfaces: A monte carlo estimator for an elliptic interface problem with nonhomogeneous flux jump conditions and a neumann boundary condition, 2025. URL https://arxiv.org/abs/2508.16767.

J. Dominguez. *Boundary elements in dynamics*. International series on computational engineering. Computational Mechanics Publications, 1993. ISBN 1853122580.

Todd F. Dupont and Yingjie Liu. Back and forth error compensation and correction methods for semi-lagrangian schemes with application to level set interface computations. *Mathematics of Computation*, 76(258):647–668, 2007. doi:10.1090/S0025-5718-06-01898-9.

Gerhard Dziuk. Finite elements for the beltrami operator on arbitrary surfaces. In *Partial Differential Equations and Calculus of Variations*, pages 142–155. Springer Berlin Heidelberg, 1988. doi:10.1007/BFb0082865.

Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.*, 26(1):4–es, January 2007. doi:10.1145/1189762.1189766.

B.S. Elepov and G.A. Mikhailov. Solution of the dirichlet problem for the equation $\delta u - cu = -q$ by a model of "walks on spheres". *USSR Computational Mathematics and Mathematical Physics*, 9(3):194–204, 1969. doi:10.1016/0041-5553(69)90070-6.

S. Ermakov and A. Sipin. The "walk in hemispheres" process and its applications to solving boundary value problems. *Vestnik St. Petersburg University: Mathematics*, 42:155–163, September 2009. doi:10.3103/S1063454109030029.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, page 15–22, 2001. doi:10.1145/383259.383260.

Bryan E. Feldman, James F. O'Brien, and Okan Arikan. Animating suspended particle explosions. *ACM Trans. Graph.*, 22(3):708–715, July 2003. doi:10.1145/882262.882336.

Bryan E. Feldman, James F. O'Brien, and Bryan M. Klingner. Animating gases with hybrid meshes. *ACM Trans. Graph.*, 24(3):904–909, July 2005. doi:10.1145/1073204.1073281.

Fan Feng, Jinyuan Liu, Shiying Xiong, Shuqi Yang, Yaorui Zhang, and Bo Zhu. Impulse fluid simulation. *IEEE Transactions on Visualization and Computer Graphics*, 29(6): 3081–3092, June 2023. doi:10.1109/TVCG.2022.3149466.

Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, page 23–30, 2001. doi:10.1145/383259.383261.

Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996. doi:10.1006/gmip.1996.0039.

Nick Foster and Dimitris Metaxas. Modeling the motion of a hot, turbulent gas. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, page 181–188, 1997. doi:10.1145/258734.258838.

Charles P. Frahm. Some novel delta-function identities. *American Journal of Physics*, 51 (9):826–829, September 1983. doi:10.1119/1.13127.

Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.*, 31(6), November 2012. doi:10.1145/2366145.2366211.

Joachim Giesen, Balint Miklos, Mark Pauly, and Camille Wormser. The scale axis transform. In *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*, page 106–115, 2009. doi:10.1145/1542362.1542388.

G. Steven Gipson. Coupling monte carlo quadrature with boundary elements to handle domain integrals in poisson type problems. *Engineering Analysis*, 2(3):138–145, 1985. doi:10.1016/0264-682X(85)90018-8.

Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Comput. Graph.*, 18 (3):213–222, January 1984. doi:10.1145/964965.808601.

L Greengard and V Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, 1987. doi:10.1016/0021-9991(87)90140-9.

Christiaan Gribble, Alexis Naveros, and Ethan Kerzner. Multi-hit ray traversal. *Journal of Computer Graphics Techniques*, 3(1):1–17, February 2014. URL `http://jcgt.org/published/0003/01/01/`.

Toshiya Hachisuka. Combined lagrangian-eulerian approach for accurate advection. In *ACM SIGGRAPH 2005 Posters*, page 114–es, 2005. doi:10.1145/1186954.1187084.

Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. A path space extension for robust light transport simulation. *ACM Trans. Graph.*, 31(6), November 2012. doi:10.1145/2366145.2366210.

David Hahn and Chris Wojtan. High-resolution brittle fracture simulation with boundary elements. *ACM Trans. Graph.*, 34(4), July 2015. doi:10.1145/2766896.

David Hahn and Chris Wojtan. Fast approximations for boundary element based brittle fracture simulation. *ACM Trans. Graph.*, 35(4), July 2016. doi:10.1145/2897824.2925902.

Francis H Harlow. The particle-in-cell method for numerical solution of problems in fluid dynamics. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), 03 1962.

Francis H. Harlow and J. Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189, 12 1965. doi:10.1063/1.1761178.

Paul Himmler and Tobias Günther. Conformal first passage for epsilon-free walk-on-spheres. *ACM Trans. Graph.*, 44(4), July 2025. doi:10.1145/3730942.

V Hnizdo. Generalized second-order partial derivatives of 1/r. *European Journal of Physics*, 32(2):287, January 2011. doi:10.1088/0143-0807/32/2/003.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. Tetrahedral meshing in the wild. *ACM Trans. Graph.*, 37(4), July 2018. doi:10.1145/3197517.3201353.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. Fast tetrahedral meshing in the wild. *ACM Trans. Graph.*, 39(4), August 2020. doi:10.1145/3386569.3392385.

Libo Huang and Dominik L. Michels. Surface-only ferrofluids. *ACM Trans. Graph.*, 39(6), November 2020. doi:10.1145/3414685.3417799.

Libo Huang, Ziyin Qu, Xun Tan, Xinxin Zhang, Dominik L. Michels, and Chenfanfu Jiang. Ships, splashes, and waves on a vast ocean. *ACM Trans. Graph.*, 40(6), December 2021. doi:10.1145/3478513.3480495.

Tianyu Huang, Jingwang Ling, Shuang Zhao, and Feng Xu. Guiding-based importance sampling for walk on stars. In *ACM SIGGRAPH 2025 Conference Papers*, 2025. doi:10.1145/3721238.3730593.

David A.B. Hyde and Ronald Fedkiw. A unified approach to monolithic solid-fluid coupling of sub-grid and more resolved solids. *Journal of Computational Physics*, 390:490–526, 2019. doi:10.1016/j.jcp.2019.03.049.

Peter Ilbery, Luke Kendall, Cyril Concolato, and Michael McCosker. Biharmonic diffusion curve images from boundary elements. *ACM Trans. Graph.*, 32(6), Nov. 2013. doi:10.1145/2508363.2508426.

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.*, 32(4), July 2013. doi:10.1145/2461912.2461916.

Pranav Jain, Ziyin Qu, Peter Yichen Chen, and Oded Stein. Neural monte carlo fluid simulation. In *ACM SIGGRAPH 2024 Conference Papers*, 2024. doi:10.1145/3641519.3657438.

Doug L. James and Dinesh K. Pai. Artdefo: accurate real time deformable objects. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, page 65–72, 1999. doi:10.1145/311535.311542.

Doug L. James and Dinesh K. Pai. Multiresolution green's function methods for interactive simulation of large-scale elastostatic objects. *ACM Trans. Graph.*, 22(1):47–82, January 2003. doi:10.1145/588272.588278.

Doug L. James, Jernej Barbič, and Dinesh K. Pai. Precomputed acoustic transfer: Output-sensitive, accurate sound generation for geometrically complex vibration sources. *ACM Trans. Graph.*, 25(3):987–995, July 2006. doi:10.1145/1141911.1141983.

Y. Le Jan and A. S. Sznitman. Stochastic cascades and 3-dimensional navier–stokes equations. *Probability Theory and Related Fields*, 109(3):343–366, 1997. doi:10.1007/s004400050135.

Wojciech Jarosz, Craig Donner, Matthias Zwicker, and Henrik Wann Jensen. Radiance caching for participating media. *ACM Trans. Graph.*, 27(1), March 2008. doi:10.1145/1330511.1330518.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Trans. Graph.*, 34(4), July 2015. doi:10.1145/2766996.

James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986. doi:10.1145/15886.15902.

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. A machine learning approach for filtering monte carlo noise. *ACM Trans. Graph.*, 34(4), July 2015. doi:10.1145/2766977.

Aneta Karaivanova, Michael Mascagni, and Nikolai A. Simonov. Parallel quasirandom walks on the boundary. *Monte Carlo Methods and Applications*, 10(3-4):311–319, December 2004. doi:10.1515/mcma.2004.10.3-4.311.

Todd Keeler and Robert Bridson. Ocean Waves Animation using Boundary Integral Equations and Explicit Mesh Tracking. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 2014. doi:10.5555/2849517.2849520.

Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A simple and robust mutation strategy for the metropolis light transport algorithm. *Comput. Graph. Forum*, 21(3):531–540, May 2002. doi:10.1111/1467-8659.t01-1-00703.

Alexander Keller. Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, page 49–56, August 1997. doi:10.1145/258734.258769.

Nathan King, Haozhe Su, Mridul Aanjaneya, Steven Ruuth, and Christopher Batty. A closest point method for pdes on manifolds with interior boundary conditions for geometry processing. *ACM Trans. Graph.*, 43(5), August 2024. doi:10.1145/3673652.

Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. A survey on sph methods in computer graphics. *Computer Graphics Forum*, 41(2):737–760, 2022. doi:10.1111/cgf.14508.

Jaroslav Krivánek, Pascal Gautron, Sumanta Pattanaik, and Kadi Bouatouch. Radiance caching for efficient global illumination computation. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):550–561, 2005. doi:10.1109/TVCG.2005.83.

Jaroslav Křivánek and Eugene d'Eon. A zero-variance-based sampling scheme for monte carlo subsurface scattering. In *ACM SIGGRAPH 2014 Talks*, pages 66:1–66:1, July 2014. doi:10.1145/2614106.2614138.

Egor Larionov, Christopher Batty, and Robert Bridson. Variational stokes: A unified pressure-viscosity solver for accurate viscous liquids. *ACM Trans. Graph.*, 36(4), July 2017. doi:10.1145/3072959.3073628.

Randall J. LeVeque. *Finite volume methods for hyperbolic problems*. Cambridge texts in applied mathematics. Cambridge University Press, 1st ed. edition, 2002. ISBN 1-107-13246-0.

Zohar Levi and Ofir Weber. On the convexity and feasibility of the bounded distortion harmonic mapping problem. *ACM Trans. Graph.*, 35(4), July 2016. doi:10.1145/2897824.2925929.

Wei Li, Kai Bai, and Xiaopei Liu. Continuous-scale kinetic fluid simulation. *IEEE Transactions on Visualization and Computer Graphics*, 25(9):2694–2709, 2018. doi:10.1109/TVCG.2018.2859931.

Wei Li, Yixin Chen, Mathieu Desbrun, Changxi Zheng, and Xiaopei Liu. Fast and scalable turbulent flow simulation with two-way coupling. *ACM Trans. Graph.*, 39(4), July 2020. doi:10.1145/3386569.3392400.

Zilu Li, Guandao Yang, Xi Deng, Christopher De Sa, Bharath Hariharan, and Steve Marschner. Neural caches for monte carlo partial differential equation solvers. In *ACM SIGGRAPH Asia 2023 Conference Papers*, December 2023. doi:10.1145/3610548.3618141.

Zilu Li, Guandao Yang, Qingqing Zhao, Xi Deng, Leonidas Guibas, Bharath Hariharan, and Gordon Wetzstein. Neural control variates with automatic integration. In *ACM SIGGRAPH 2024 Conference Papers*, 2024. doi:10.1145/3641519.3657395.

Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Trans. Graph.*, 27(3):1–10, August 2008. doi:10.1145/1360612.1360677.

Yijun Liu, Subrata Mukherjee, Naoshi Nishimura, Martin Schanz, Wenjing Ye, Alok Sutradhar, Ernie Pan, Ney Augusto Dumont, Attilio Frangi, and Andres Saez. Recent Advances and Emerging Applications of the Boundary Element Method. *Applied Mechanics Reviews*, 64(3), March 2012. doi:10.1115/1.4005491.

Frank Losasso, Tamar Shinar, Andrew Selle, and Ronald Fedkiw. Multiple interacting liquids. *ACM Trans. Graph.*, 25(3):812–819, July 2006. doi:10.1145/1141911.1141960.

Thomas Lundgren and Petros Koumoutsakos. On the generation of vorticity at a free surface. *Journal of Fluid Mechanics*, 382:351–366, 1999. doi:10.1017/S0022112098003978.

Chaoyang Lyu, Wei Li, Mathieu Desbrun, and Xiaopei Liu. Fast and versatile fluid-solid coupling for turbulent flow simulation. *ACM Trans. Graph.*, 40(6), December 2021. doi:10.1145/3478513.3480493.

Jaehwan Ma, Sang Won Bae, and Sunghee Choi. 3d medial axis point approximation using nearest neighbors and the normal field. *The Visual Computer*, 28(1):7–19, 2012. doi:10.1007/s00371-011-0594-7.

Miles Macklin, Kier Storey, Michelle Lu, Pierre Terdiman, Nuttapong Chentanez, Stefan Jeschke, and Matthias Müller. Small steps in physics simulation. In *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2019. doi:10.1145/3309486.3340247.

Richard Henri MacNeal. *The solution of partial differential equations by means of electrical networks*. PhD thesis, California Institute of Technology, 1949.

Abhishek Madan, Nicholas Sharp, Francis Williams, Ken Museth, and David I.W. Levin. Stochastic barnes-hut approximation for fast summation on the gpu. In *ACM SIGGRAPH 2025 Conference Papers*, 2025. doi:10.1145/3721238.3730725.

Gisiro Maruyama. Continuous markov processes and stochastic equations. *Rendiconti del Circolo Matematico di Palermo*, 4(1):48–90, 1955.

Thomas März and Colin B. Macdonald. Calculus on surfaces with general closest point functions. *SIAM Journal on Numerical Analysis*, 50(6):3303–3328, 2012. doi:10.1137/120865537.

Michael Mascagni. Monte carlo methods for partial differential equations. https://www.cs.fsu.edu/~mascagni/mcpdenew.pdf, 2011. Talk slides, Florida State University.

Michael Mascagni and Nikolai A. Simonov. Monte carlo methods for calculating some physical properties of large molecules. *SIAM Journal on Scientific Computing*, 26(1): 339–357, 2004. doi:10.1137/S1064827503422221.

Olivier Mercier, Xi-Yuan Yin, and Jean-Christophe Nave. The characteristic mapping method for the linear advection of arbitrary sets, 2013. URL https://arxiv.org/abs/1309.2731.

Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247), 1949.

Balint Miklos, Joachim Giesen, and Mark Pauly. Discrete scale axis representations for 3d geometry. In *ACM SIGGRAPH 2010 Papers*, 2010. doi:10.1145/1833349.1778838.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. Boundary value caching for walk on spheres. *ACM Trans. Graph.*, 42(4), August 2023. doi:10.1145/3592400.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. Differential walk on spheres. *ACM Trans. Graph.*, 43(6), November 2024a. doi:10.1145/3687913.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. Walkin' robin: Walk on stars with robin boundary conditions. *ACM Trans. Graph.*, 43(4), July 2024b. doi:10.1145/3658153.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. Solving partial differential equations in participating media. *ACM Trans. Graph.*, 44(4), July 2025. doi:10.1145/3731152.

Zackary Misso, Benedikt Bitterli, Iliyan Georgiev, and Wojciech Jarosz. Unbiased and consistent rendering using biased estimators. *ACM Trans. Graph.*, 41(4), July 2022. doi:10.1145/3528223.3530160.

Marek Krzysztof Misztal, Kenny Erleben, Adam Bargteil, Jens Fursund, Brian Bunch Christensen, Jakob Andreas Bærentzen, and Robert Bridson. Multiphase flow of immiscible fluids on unstructured moving meshes. *IEEE Transactions on Visualization and Computer Graphics*, 20(1):4–16, 2014. doi:10.1109/TVCG.2013.97.

Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. Energy-preserving integrators for fluid animation. *ACM Trans. Graph.*, 28(3), July 2009. doi:10.1145/1531326.1531344.

Matthias Müller and Markus Gross. Interactive virtual materials. In *Proceedings of Graphics Interface 2004*, page 239–246, 2004. doi:10.5555/1006058.1006087.

Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 49–54, 2002. doi:10.1145/545261.545269.

Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 154–159, 2003. doi:10.5555/846276.846298.

Mervin E. Muller. Some Continuous Monte Carlo Methods for the Dirichlet Problem. *The Annals of Mathematical Statistics*, 27(3):569 – 589, 1956. doi:10.1214/aoms/1177728169.

Thomas Müller, Markus Gross, and Jan Novák. Practical path guiding for efficient light-transport simulation. *Computer Graphics Forum*, 36(4):91–100, 2017. doi:10.1111/cgf.13227.

Mohammad Sina Nabizadeh, Ravi Ramamoorthi, and Albert Chern. Kelvin transformations for simulations on infinite domains. *ACM Trans. Graph.*, 40(4), July 2021. doi:10.1145/3450626.3459809.

Mohammad Sina Nabizadeh, Stephanie Wang, Ravi Ramamoorthi, and Albert Chern. Covector fluids. *ACM Trans. Graph.*, 41(4), July 2022. doi:10.1145/3528223.3530120.

Mohammad Sina Nabizadeh, Ritoban Roy-Chowdhury, Hang Yin, Ravi Ramamoorthi, and Albert Chern. Fluid implicit particles on coadjoint orbits. *ACM Trans. Graph.*, 43(6), November 2024. doi:10.1145/3687970.

Hong Chul Nam, Julius Berner, and Anima Anandkumar. Solving poisson equations using neural walk-on-spheres. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024. doi:10.5555/3692070.3693583.

Rahul Narain, Jonas Zehnder, and Bernhard Thomaszewski. A second-order advection-reflection solver. *Proc. ACM Comput. Graph. Interact. Tech.*, 2(2), July 2019. doi:10.1145/3340257.

Xingyu Ni, Ruicheng Wang, Bin Wang, and Baoquan Chen. An induce-on-boundary magnetostatic solver for grid-based ferrofluids. *ACM Trans. Graph.*, 43(4), July 2024. doi:10.1145/3658124.

Michael B. Nielsen and Robert Bridson. Guide shapes for high resolution naturalistic liquid simulation. *ACM Trans. Graph.*, 30(4), July 2011. doi:10.1145/2010324.1964978.

Tomoyuki Nishita and Eihachiro Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. *SIGGRAPH Comput. Graph.*, 19(3):23–30, July 1985. doi:10.1145/325165.325169.

Jan Novák, Iliyan Georgiev, Johannes Hanika, and Wojciech Jarosz. Monte carlo methods for volumetric light transport simulation. *Computer Graphics Forum*, 37(2):551–576, 2018. doi:10.1111/cgf.13383.

Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. *ACM Trans. Graph.*, 27(3):1–8, August 2008. doi:10.1145/1360612.1360691.

Art B. Owen. *Monte Carlo theory, methods and examples.* https://artowen.su.domains/mc/, 2013.

Sang Il Park and Myoung Jun Kim. Vortex fluid for gaseous phenomena. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 261–270, 2005. doi:10.1145/1073368.1073406.

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: A general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4), July 2010. doi:10.1145/1778765.1778803.

Clemens Pechstein. Special lecture on boundary element methods, 2013. URL https://www.numa.uni-linz.ac.at/Teaching/LVA/2013s/SpezVO/bem13.pdf.

M. F. Peeters, W. G. Habashi, and E. G. Dueck. Finite element stream function-vorticity solutions of the incompressible navier-stokes equations. *International Journal for Numerical Methods in Fluids*, 7(1):17–27, 1987. doi:10.1002/fld.1650070103.

P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990. doi:10.1109/34.56205.

A. Petras and S.J. Ruuth. Pdes on moving surfaces via the closest point method and a modified grid based particle method. *Journal of Computational Physics*, 312:139–156, 2016. doi:10.1016/j.jcp.2016.02.024.

Tobias Pfaff, Nils Thuerey, and Markus Gross. Lagrangian vortex sheets for animating fluids. *ACM Trans. Graph.*, 31(4), July 2012. doi:10.1145/2185520.2185608.

Matt Pharr. Guest editor's introduction: Special issue on production rendering. *ACM Trans. Graph.*, 37(3), July 2018. doi:10.1145/3212511.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, Burlington, Massachusetts, 2018.

Cécile Piret. The orthogonal gradients method: A radial basis functions method for solving partial differential equations on arbitrary surfaces. *Journal of Computational Physics*, 231(14):4662–4675, 2012. doi:10.1016/j.jcp.2012.03.007.

H. Power and L.C. Wrobel. *Boundary Integral Methods in Fluid Mechanics*. Computational Mechanics Publications, 1995. ISBN 1562521764.

Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Trans. Graph.*, 21(3):703–712, July 2002. doi:10.1145/566654.566640.

Yang Qi, Dario Seyb, Benedikt Bitterli, and Wojciech Jarosz. A bidirectional formulation for walk on spheres. *Comput. Graph. Forum*, 41(4):51–62, 2022. doi:10.1111/cgf.14586.

Ziyin Qu, Xinxin Zhang, Ming Gao, Chenfanfu Jiang, and Baoquan Chen. Efficient and conservative fluids using bidirectional mapping. *ACM Trans. Graph.*, 38(4), July 2019. doi:10.1145/3306346.3322945.

Inigo Quilez and Pol Jeremias. Shadertoy. https://www.shadertoy.com, 2013.

J.N. Reddy. *An introduction to the finite element method*. McGraw-Hill series in mechanical engineering. McGraw-Hill, 2nd ed. edition, 1993. ISBN 0070513554.

Damien Rioux-Lavoie, Ryusuke Sugimoto, Tümay Özdemir, Naoharu H. Shimada, Christopher Batty, Derek Nowrouzezahrai, and Toshiya Hachisuka. A monte carlo method for fluid simulation. *ACM Trans. Graph.*, 41(6), November 2022. doi:10.1145/3550454.3555450.

Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag, 1999.

Steven J. Ruuth and Barry Merriman. A simple embedding method for solving partial differential equations on surfaces. *J. Comput. Phys.*, 227(3):1943–1961, January 2008. doi:10.1016/j.jcp.2007.10.009.

Youcef Saad and Martin H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. doi:10.1137/0907058.

Karl Sabelfeld and Anastasya Kireeva. A new global random walk algorithm for calculation of the solution and its derivatives of elliptic equations with constant coefficients in an arbitrary set of points. *Applied Mathematics Letters*, 107:106466, 2020. doi:10.1016/j.aml.2020.106466.

Karl K. Sabelfeld. Vector algorithms in the monte-carlo method for solving systems of second-order elliptic equations and lame's equation. *Doklady Akademii Nauk SSSR*, 262 (5):1076–1080, 1982. URL https://www.mathnet.ru/eng/dan45069. (In Russian).

Karl K. Sabelfeld. *Monte Carlo Methods: in Boundary Value Problems*. Scientific Computation. Springer, Berlin, Heidelberg, 1991.

Karl K. Sabelfeld. Stochastic boundary methods of fundamental solutions for solving pdes. *Engineering Analysis with Boundary Elements*, 36(7):1092–1103, July 2012. doi:10.1016/j.enganabound.2012.02.003.

Karl K. Sabelfeld and Georgy Agarkov. Randomized vector algorithm with iterative refinement for solving boundary integral equations. *Monte Carlo Methods and Applications*, 30(4):375–388, 2024. doi:10.1515/mcma-2024-2022.

Karl K. Sabelfeld and Georgy Agarkov. Combining randomized and deterministic iterative algorithms for high accuracy solution of large linear systems and boundary integral equations. *Monte Carlo Methods and Applications*, 31(2):145–162, 2025. doi:10.1515/mcma-2025-2008.

Karl K. Sabelfeld and Nikolai A. Simonov. *Random Walks on Boundary for Solving PDEs*. De Gruyter, Berlin, 1994. doi:10.1515/9783110942026.

Takahiro Sato, Christopher Batty, Takeo Igarashi, and Ryoichi Ando. Spatially adaptive long-term semi-lagrangian method for accurate velocity advection. *Computational Visual Media*, 4(3):223–230, 2018. doi:10.1007/s41095-018-0117-9.

Rohan Sawhney. *Monte Carlo Geometry Processing: A Grid-Free Approach to Solving Partial Differential Equations on Volumetric Domains*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2024.

Rohan Sawhney and Keenan Crane. Monte carlo geometry processing: a grid-free approach to pde-based methods on volumetric domains. *ACM Trans. Graph.*, 39(4), August 2020. doi:10.1145/3386569.3392374.

Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. Grid-free monte carlo for pdes with spatially varying coefficients. *ACM Trans. Graph.*, 41(4), July 2022. doi:10.1145/3528223.3530134.

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 Trans. Graph.*, 42(4), August 2023. doi:10.1145/3592398.

Martin Schanz. *Wave propagation in viscoelastic and poroelastic continua: a boundary element approach*, volume 2. Springer, 2001. doi:10.1007/978-3-540-44575-3.

Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph.*, 24(3):910–914, July 2005. doi:10.1145/1073204.1073282.

Andrew Selle, Ronald Fedkiw, ByungMoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2): 350–371, 2008. doi:10.1007/s10915-007-9166-4.

Irina Shalimova and Karl K. Sabelfeld. A meshfree random walk on boundary algorithm with iterative refinement. *Monte Carlo Methods and Applications*, 31(2):131–143, 2025. doi:10.1515/mcma-2025-2007.

Nicholas Sharp and Keenan Crane. A laplacian for nonmanifold triangle meshes. *Computer Graphics Forum*, 39(5):69–80, 2020. doi:10.1111/cgf.14069.

Nicholas Sharp, Keenan Crane, et al. Geometrycentral: A modern c++ library of data structures and algorithms for geometry processing, 2019. URL `https://geometry-central.net/`. Computer Software.

N. H. Shimada and T. Hachisuka. Quantum coin method for numerical integration. *Computer Graphics Forum*, 39(6):243–257, 2020. doi:10.1111/cgf.14015.

Side Effects Software Inc. Houdini 20.5, 2024. URL `https://www.sidefx.com/docs/houdini/`. Computer Software.

Nikolai A Simonov. Walk-on-spheres algorithm for solving boundary-value problems with continuity flux conditions. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 633–643, 2008. doi:10.1007/978-3-540-74496-2_38.

Nikolai A Simonov. Walk-on-spheres algorithm for solving third boundary value problem. *Applied Mathematics Letters*, 64:156–161, February 2017. doi:10.1016/j.aml.2016.09.008.

Justin Solomon, Amir Vaxman, and David Bommes. Boundary element octahedral fields in volumes. *ACM Trans. Graph.*, 36(4), May 2017. doi:10.1145/3072959.3065254.

Silei Song, Arash Fahim, and Michael Mascagni. Wosnn: Stochastic solver for pdes with machine learning, 2025. URL `https://arxiv.org/abs/2509.00204`.

Jos Stam. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, page 121–128, 1999. doi:10.1145/311535.311548.

Jos Stam. Flows on surfaces of arbitrary topology. *ACM Trans. Graph.*, 22(3):724–731, July 2003. doi:10.1145/882262.882338.

Ryusuke Sugimoto, Christopher Batty, and Toshiya Hachisuka. Surface-only dynamic deformables using a boundary element method. *Computer Graphics Forum*, 41(8):75–86, 2022. doi:10.1111/cgf.14625.

Ryusuke Sugimoto, Terry Chen, Yiti Jiang, Christopher Batty, and Toshiya Hachisuka. A practical walk-on-boundary method for boundary value problems. *ACM Trans. Graph.*, 42(4), July 2023. doi:10.1145/3592109.

Ryusuke Sugimoto, Christopher Batty, and Toshiya Hachisuka. Velocity-based monte carlo fluids. In *ACM SIGGRAPH 2024 Conference Papers*, 2024a. doi:10.1145/3641519.3657405.

Ryusuke Sugimoto, Nathan King, Toshiya Hachisuka, and Christopher Batty. Projected walk on spheres: A monte carlo closest point method for surface pdes. In *SIGGRAPH Asia 2024 Conference Papers*, SA '24, 2024b. doi:10.1145/3680528.3687599.

Ryusuke Sugimoto, Jeff Lait, Christopher Batty, and Toshiya Hachisuka. Galerkin method of regularized stokeslets for procedural fluid flow with control curves. In *SIGGRAPH Asia 2024 Technical Communications*, 2024c. doi:10.1145/3681758.3698019.

Timothy Sun, Papoj Thamjaroenporn, and Changxi Zheng. Fast multipole representation of diffusion curves and points. *ACM Trans. Graph.*, 33(4), July 2014. doi:10.1145/2601097.2601187.

Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong, and Baining Guo. Diffusion curve textures for resolution independent texture mapping. *ACM Trans. Graph.*, 31(4), July 2012. doi:10.1145/2185520.2185570.

Andrea Tagliasacchi, Thomas Delame, Michela Spagnuolo, Nina Amenta, and Alexandru Telea. 3d skeletons: A state-of-the-art report. *Computer Graphics Forum*, 35(2):573–597, 2016. doi:10.1111/cgf.12865.

Justin Talbot, David Cline, and Parris K. Egbert. Importance resampling for global illumination. In *Proceedings of the Eurographics Symposium on Rendering Techniques*, pages 139–146. Eurographics Association, 2005. doi:10.2312/EGWR/EGSR05/139-146.

Geoffrey Ingram Taylor and Albert Edward Green. Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences*, 158(895):499–521, 1937.

Jerry Tessendorf. Simulating ocean water. *Simulating nature: realistic and interactive techniques. SIGGRAPH*, 1(2):5, 2004.

Jerry Tessendorf and Brandon Pelfrey. The characteristic map for fast and efficient vfx fluid simulations. In *Computer Graphics International Workshop on VFX, Computer Animation, and Stereo Movies*, Ottawa, Canada, 2011.

J. W. Thomas. *Numerical partial differential equations : finite difference methods*, volume 22 of *Texts in applied mathematics*. Springer, 1st ed. 1995. edition, 1995. ISBN 1-4899-7278-1.

Xingze Tian and Tobias Günther. Monte Carlo Methods for 2D Flow Visualization. In *EuroVis 2025 - Short Papers*. The Eurographics Association, 2025. doi:10.2312/evs.20251089.

Xiaochun Tong and Toshiya Hachisuka. Practical stylized nonlinear monte carlo rendering. In *ACM SIGGRAPH 2025 Conference Papers*, 2025. doi:10.1145/3721238.3730686.

Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *SIGGRAPH Comput. Graph.*, 25(4):289–298, July 1991. doi:10.1145/127719.122749.

Nobuyuki Umetani, Athina Panotopoulou, Ryan Schmidt, and Emily Whiting. Printone: interactive resonance simulation for free-form print-wind instrument design. *ACM Trans. Graph.*, 35(6), December 2016. doi:10.1145/2980179.2980250.

Jasper van de Gronde. A high quality solver for diffusion curves. Master's thesis, University of Groningen, 2010. URL `https://fse.studenttheses.ub.rug.nl/id/eprint/9496`.

Adriaan van Wijngaarden. A transformation of formal series. *Indagationes mathematicae*, 15:522–543, January 1953.

Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, Stanford, CA, USA, 1998.

Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 419–428. ACM, 1995. doi:10.1145/218380.218498.

Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, page 65–76, 1997. doi:10.1145/258734.258775.

Ingrid von Glehn, Thomas März, and Colin B. Macdonald. An embedded method-of-lines approach to solving partial differential equations on surfaces, 2013. URL https://arxiv.org/abs/1307.5657.

Jiří Vorba and Jaroslav Křivánek. Adjoint-driven russian roulette and splitting in light transport simulation. *ACM Trans. Graph.*, 35(4), July 2016. doi:10.1145/2897824.2925912.

Ingo Wald. Owl - the optix 7 wrapper library, 2023. URL http://owl-project.github.io. Computer Software.

He Wang, Kirill A. Sidorov, Peter Sandilands, and Taku Komura. Harmonic parameterization by electrostatics. *ACM Trans. Graph.*, 32(5), October 2013. doi:10.1145/2503177.

Xiaokun Wang, Yanrui Xu, Sinuo Liu, Bo Ren, Jiří Kosinka, Alexandru C. Telea, Jiamin Wang, Chongming Song, Jian Chang, Chenfeng Li, Jian Jun Zhang, and Xiaojuan Ban. Physics-based fluid simulation in computer graphics: Survey, research trends, and challenges. *Computational Visual Media*, 10(5):803–858, 2024. doi:10.1007/s41095-023-0368-y.

Gregory J Ward, Francis M Rubinstein, and Robert D Clear. A ray tracing solution for diffuse interreflection. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 85–92, 1988.

Rex West and Sayan Mukherjee. Stylized rendering as a function of expectation. *ACM Trans. Graph.*, 43(4), July 2024. doi:10.1145/3658161.

Guofu Xie, Xin Sun, Xin Tong, and Derek Nowrouzezahrai. Hierarchical diffusion curves for accurate automatic image vectorization. *ACM Trans. Graph.*, 33(6), Nov. 2014. doi:10.1145/2661229.2661275.

Shiying Xiong, Rui Tao, Yaorui Zhang, Fan Feng, and Bo Zhu. Incompressible flow simulation on vortex segment clouds. *ACM Trans. Graph.*, 40(4), July 2021. doi:10.1145/3450626.3459865.

Shiying Xiong, Zhecheng Wang, Mengdi Wang, and Bo Zhu. A clebsch method for free-surface vortical flow simulation. *ACM Trans. Graph.*, 41(4), July 2022. doi:10.1145/3528223.3530150.

Dongbin Xiu and George Em Karniadakis. A semi-lagrangian high-order method for navier–stokes equations. *Journal of Computational Physics*, 172(2):658–684, 2001. doi:10.1006/jcph.2001.6847.

Shuqi Yang, Shiying Xiong, Yaorui Zhang, Fan Feng, Jinyuan Liu, and Bo Zhu. Clebsch gauge fluid. *ACM Trans. Graph.*, 40(4), July 2021. doi:10.1145/3450626.3459866.

Xingyu Ye, Xiaokun Wang, Yanrui Xu, Jiří Kosinka, Alexandru C. Telea, Lihua You, Jian Jun Zhang, and Jian Chang. Monte carlo vortical smoothed particle hydrodynamics for simulating turbulent flows. *Computer Graphics Forum*, 43(2), 2024. doi:10.1111/cgf.15024.

Ekrem Fatih Yilmazer, Delio Vicini, and Wenzel Jakob. Solving inverse pde problems using grid-free monte carlo estimators. *ACM Trans. Graph.*, 43(6), November 2024. doi:10.1145/3687990.

Hang Yin, Mohammad Sina Nabizadeh, Baichuan Wu, Stephanie Wang, and Albert Chern. Fluid cohomology. *ACM Trans. Graph.*, 42(4), July 2023. doi:10.1145/3592402.

Zihan Yu, Lifan Wu, Zhiqian Zhou, and Shuang Zhao. A differential monte carlo solver for the poisson equation. In *ACM SIGGRAPH 2024 Conference Papers*, 2024. doi:10.1145/3641519.3657460.

Jonas Zehnder, Rahul Narain, and Bernhard Thomaszewski. An advection-reflection solver for detail-preserving fluid simulation. *ACM Trans. Graph.*, 37(4), July 2018. doi:10.1145/3197517.3201324.

Xinxin Zhang and Robert Bridson. A pppm fast summation method for fluids and beyond. *ACM Trans. Graph.*, 33(6), November 2014. doi:10.1145/2661229.2661261.

Xinxin Zhang, Robert Bridson, and Chen Greif. Restoring the missing vorticity in advection-projection fluid solvers. *ACM Trans. Graph.*, 34(4), July 2015. doi:10.1145/2766982.

Changxi Zheng and Doug L. James. Harmonic fluids. *ACM Trans. Graph.*, 28(3), July 2009. doi:10.1145/1531326.1531343.

Changxi Zheng and Doug L. James. Rigid-body fracture sound with precomputed soundbanks. *ACM Trans. Graph.*, 29(4), July 2010. doi:10.1145/1778765.1778806.

Huan-Xiang Zhou, Attila Szabo, Jack F. Douglas, and Joseph B. Hubbard. A brownian dynamics algorithm for calculating the hydrodynamic friction and the electrostatic capacitance of an arbitrarily shaped object. *The Journal of Chemical Physics*, 100(5): 3821–3826, March 1994. doi:10.1063/1.466371.

Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24 (3):965–972, July 2005. doi:10.1145/1073204.1073298.

Yufeng Zhu, Robert Bridson, and Chen Greif. Simulating rigid body fracture with surface meshes. *ACM Trans. Graph.*, 34(4), July 2015. doi:10.1145/2766942.