Solving Inverse PDE Problems using Grid-Free Monte Carlo Estimators

EKREM FATIH YILMAZER, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
DELIO VICINI, Google Inc., Switzerland
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Fig. 1. We apply our inverse PDE solver to a 2D electrical impedance tomography experiment [Hauptmann et al. 2017], in which an electric current flows through a saline-filled water tank containing conducting objects of different sizes (photographs in middle). The marker in the middle indicates the center. Injecting a current using 2 of 16 uniformly spaced electrodes at the tank boundary generates a measurable voltage difference at the other electrodes, and injecting at various locations produces a matrix \( u_{\text{ref}} \) of measurements. The objective of this inverse problem is to infer the properties of the conductor from this data. We perform a differentiable simulation of this setup to optimize the center and radius of a conducting circle. The frames on the left show the progression of the optimization (columns \( a \)), while the rightmost two columns reveal how the predicted voltages become increasingly consistent with the measurement (columns \( d \)).

Partial differential equations can model diverse physical phenomena including heat diffusion, incompressible flows, and electrostatic potentials. Given a description of an object’s boundary and interior, traditional methods solve such PDEs by densely meshing the interior and then solving a large and sparse linear system derived from this mesh. Recent grid-free solvers take an alternative approach and avoid this complexity in exchange for randomness: they compute stochastic solution estimates and generally bear a striking resemblance to physically-based rendering algorithms.

In this article, we develop algorithms targeting the inverse form of this problem: given an already existing solution of a PDE, we infer parameters characterizing the boundary and interior. In the grid-free setting, there are again significant connections to rendering, and we show how insights from both fields can be combined to compute unbiased derivative estimates that enable gradient-based optimization. In this process, we encounter new challenges that must be addressed to obtain practical solutions. We introduce acceleration and variance reduction strategies and show how to differentiate branching random walks in reverse mode.

We finally demonstrate our approach on both simulated data and a real-world electrical impedance tomography experiment, where we reconstruct the position of a conducting object from voltage measurements taken in a saline-filled tank.

CCS Concepts:

- Mathematics of computing → Partial differential equations
- Computing methodologies → Rendering

Additional Key Words and Phrases: Walk on Spheres, Differentiable Rendering, Path Replay Backpropagation, Electrical Impedance Tomography

ACM Reference Format:

Ekrem Fatih Yilmazer, Delio Vicini, and Wenzel Jakob. 2024. Solving Inverse PDE Problems using Grid-Free Monte Carlo Estimators. ACM Trans. Graph. 43, 6, Article 175 (December 2024), 18 pages. https://doi.org/10.1145/3687990

1 INTRODUCTION

Many physical phenomena are naturally described using partial differential equations (PDEs). For example, the heat equation models the spread of thermal energy in a potentially heterogeneous material. Solvers that numerically approximate solutions of such PDEs are in widespread use. We pursue the opposite direction in this article, which is known as an inverse PDE problem: estimating unknown parameters from observations of the solution. This set of unknown parameters could include various PDE coefficients, boundary conditions, and even the shape of the domain.
Such problems arise in diverse scientific and engineering contexts, for example to determine the physical parameters of a thermal conductor from measurements [Cannon 1964]. Electrical impedance tomography [Cheney et al. 1999] seeks to reconstruct the interior of a living organism. Electrodes provide measurements of the electric field, which is influenced by the tissue’s conductivity, impedance, and dielectric permittivity.

Our approach entails differentiating the solver and recovering the unknown parameters using gradient descent. However, one issue with conventional PDE solvers based on the finite element method (FEM) is that they require a meshing step that can be fragile and computationally costly. An alternative is using Monte Carlo PDE solvers based on the walk on spheres (WoS) [Muller 1956]. These grid-free methods sample random paths in the domain to compute unbiased estimates of the solution. Grid-free solvers have recently attracted significant attention in the computer graphics community, partly owing to the similarities to rendering [Sawhney and Crane 2020] and the algorithmic synergies that this creates [Sawhney et al. 2022, 2023; Qi et al. 2022; Bati et al. 2023].

A common issue with gradient-based optimization is that the standard approach for reverse-mode differentiation (also known as backpropagation) reverses the data dependencies of an underlying computation. When applied to the WoS algorithm, this means that intermediate results of a large number of iterations would need to be stored to enable the subsequent differentiation.

In rendering, recent progress has led to the development of differentiable rendering methods [Gkioulekas et al. 2013; Li et al. 2018; Nimier-David et al. 2019] that estimate derivatives of complete light transport simulations. A similar issue arises here as well: light paths can potentially be very long, particularly in highly-scattering media, which makes naive differentiation prohibitively memory-intensive. This has sparked work on specialized adjoint methods [Nimier-David et al. 2020; Vicini et al. 2021] that cast differentiation into an independent simulation of “derivative light” to avoid these costs.

We demonstrate that this idea also applies to Monte Carlo PDE solvers, where it leads to unbiased derivative estimators for several variants and extensions of the WoS algorithm. At the same time, unique characteristics of this setting require the development of new solutions: for example, PDEs with spatially variable coefficients often require a variance reduction strategy where random walks can split. We extend Path Replay Backpropagation [Vicini et al. 2021] to branching random walks to support this requirement.

In our experiments, we found that the forward simulation was often the main computational bottleneck, and we propose several improvements to accelerate convergence of both primal and adjoint simulations: first, to better handle PDEs with a heterogeneous diffusion coefficient, we introduce an adaptive sphere radius selection. Second, we develop a fast way to evaluate and sample the Green’s function, accelerating convergence by up to an order of magnitude.

As an application of our work, we investigate electrical impedance tomography (EIT) [Cheney et al. 1999]. Our method provides great flexibility in the parameterization of the unknown conductance sought by EIT that we demonstrate in recovering continuously varying and binary conductance fields. Finally, we use our solver to reconstruct the shape and position of a conducting object in a saline-filled tank (Figure 1). This demonstrates that our approach can translate to real-world data despite modeling approximations.

In summary, our contributions are:

- We develop efficient adjoints for the Walk on Spheres [Sawhney and Crane 2020] and Walk on Stars [Sawhney et al. 2023], demonstrating recovery of governing parameters (e.g., diffusion coefficient, source terms).
- We then extend these methods to branching random walks.
- We improve the performance of the forward PDE solvers by introducing an adaptive sphere radius selection and optimized Green’s function evaluation.
- We apply these ideas to electrical impedance tomography and present results on measured data.

The experiments in this paper use 2D domains for ease of visualization, though the presented ideas straightforwardly generalize to higher dimensions. We will release an open source implementation of our method to ensure reproducibility.

2 RELATED WORK

We briefly review prior work on Monte Carlo PDE solvers, differentiable Monte Carlo estimators and inverse PDE problems.

Monte Carlo PDE solvers. The solvers in this paper all build on variations of walk on spheres [Muller 1956]. Sawhney and Crane [2020] were the first to explore this technique in context of computer graphics, realizing their relationship to existing Monte Carlo rendering methods [Pharr et al. 2020]. Unique characteristics of these solvers are that they provide statistically unbiased solution estimates with a computation that is local and progressive in nature, all without requiring a mesh of the domain’s interior. This makes it possible to solve and even interactively preview problems on large and complex domains. The base algorithm was later extended to spatially-varying coefficients [Sawhney et al. 2022], as well as Neumann [Sawhney et al. 2023] and Robin boundary conditions [Miller et al. 2024]. Other recent uses of Monte Carlo PDE solvers include estimators for fluid motion [Li et al. 2023] and coupled solvers for thermal transport [Bati et al. 2023].

As in Monte Carlo rendering, the stochastic nature of estimates motivates specialized variance reduction strategies. Qi et al. [2022] propose a bidirectional estimator for PDEs with sparse source terms. Other works cache and reuse evaluations on the boundary [Li et al. 2023] or interior [Bakbouk 2023] or approximate the solution using a neural cache [Li et al. 2023]. Nabizadeh et al. [2021] warp space inside-out to efficiently solve PDEs on unbounded domains.

Differentiable Monte Carlo methods. Given the origins of the Monte Carlo method in the area of nuclear physics, it is of no surprise that the first uses of differential Monte Carlo likewise investigated nuclear processes, for example to determine the efficiency of neutron breeding [Mikhailov 1967], design radiation shielding [Braimina et al. 1967], or model the criticality transition in nuclear reactors [Lux and Koblinger 1990]. In computer graphics, the use of differential Monte Carlo was initially focused on appearance estimation [Gkioulekas et al. 2013, 2016; Khungurn et al. 2015; Velinov et al. 2018]. Recently, general algorithms [Li et al. 2018] and systems [Nimier-David et al. 2019; Jakob et al. 2022] have enabled more systematic use of differentiable Monte Carlo simulations. In rendering, special consideration
is required to differentiate visibility discontinuities [Li et al. 2018; Loubet et al. 2019; Zhang et al. 2020; Vicini et al. 2022]. Concurrently to our work, Yu et al. [2024] discuss application of some of those ideas to optimize the boundary shape of Poisson problems. The non-trivial control flow and high number of operations found in rendering algorithms make conventional automatic differentiation impractical, which has motivated the development of more efficient adjoint methods [Nimier-David et al. 2020; Vicini et al. 2021] that we also build upon.

Inverse PDE problems. PDE-constrained inverse problems occur in a wide range of practical applications such as electrical impedance tomography [Barber and Brown 1984; Adler and Boyle 2017], fluid control [McNamara et al. 2004], geophysics [Mosser et al. 2020], nanophotonics [Molesky et al. 2018] and thermal design [França et al. 2003]. In computer graphics, Zhao et al. [2018] solve an inverse problem to control diffusion curves [Orzan et al. 2013].

Electrical impedance tomography. As an example application, we demonstrate the use of our solvers for electrical impedance tomography. Prior work can be categorized into statistical and deterministic approaches. Statistical methods are mainly based on Bayesian inversion [Liu et al. 2018, 2020a,b]. Deterministic approaches either assume linearity [Santosa and Vogelius 1990; Barber and Brown 1984; Calderón 2006; Cheney et al. 1990; Adler et al. 2009] or use iterative algorithms such as Gauss-Newton or Newton-Raphson [Bayford 2006]. An example of a direct non-linear method is the D-Bar algorithm [Hamilton et al. 2012, 2018]. Other approaches use level set methods [Liu et al. 2017], morphable components [Liu and Du 2021] or directly predict conductivities using neural networks [Guardo et al. 1991; Li et al. 2017; Zhang et al. 2022].

3 BACKGROUND: MONTE CARLO PDE SOLVERS

The discussion of our method (Section 4) assumes familiarity with Monte Carlo PDE solvers that we review here for completeness. Table 1 lists notation used throughout the paper. For a more thorough introduction, we also refer to the original publications [Sawhney and Crane 2020; Sawhney et al. 2022, 2023].

3.1 Walk on Spheres

Screened Poisson equation. We consider the screened Poisson equation with Dirichlet boundary conditions as an introductory problem:

\begin{equation}
\Delta u(x) - \sigma u(x) = -f(x) \quad x \in \Omega,
\end{equation}

\begin{equation}
u(x) = g(x) \quad x \in \partial \Omega.
\end{equation}

Here, \(u(x)\) denotes the solution at a point inside the domain \(\Omega\). The function \(f(x)\) is a spatially-varying source term, and \(g(x)\) prescribes the solution values on the domain boundary \(\partial \Omega\). The scalar screening coefficient \(\sigma\) models a decay in the influence of distant sources and boundaries. Solutions to Equation (1) satisfy a generalized mean value property [Muller 1956; Sawhney and Crane 2020] that expresses the solution \(u(x)\) in the form of an integral equation

\begin{equation}
u(x) = \int_{\partial B(x)} P_\sigma(x, x') u(x') \, dx' + \int_{B(x)} G_\sigma(x, y) f(y) \, dy.
\end{equation}

Table 1. Notation used throughout paper

| Term | Interpretation |
|------|----------------|
| \(\Omega, \partial \Omega\) | The domain and its boundary |
| \(B, \partial B\) | The current ball and its boundary |
| \(\partial B\) | The surface area of the ball |
| \(S_t, \partial S_t\) | The current ball and its boundary |
| \(\partial \Omega_d\) | Part of \(\partial \Omega\) with Dirichlet boundary conditions |
| \(\partial \Omega_n\) | Part of \(\partial \Omega\) with Neumann boundary conditions |
| \(\partial S_t\) | Part of \(\partial S_t\) with Neumann boundary conditions |
| \(f(y)\) | Spatially varying source term at \(x \in \Omega\) |
| \(g(x)\) | Value of the solution at \(x \in \partial \Omega_d\) |
| \(h(x)\) | Normal derivative of the solution at \(x \in \partial \Omega_d\) |
| \(\sigma(y)\) | Spatially varying screening coefficient |
| \(\alpha(y)\) | Spatially varying diffusion coefficient |
| \(x, x'\) | Current and next vertex of the Walk on Spheres |
| \(P_\sigma, G_\sigma\) | Poisson kernel and Green’s function |

where \(B(x) \subset \Omega\) is a ball around \(x\), \(P_\sigma(x, x')\) is the Poisson kernel and \(G_\sigma(x, y)\) the Green’s function. The Poisson kernel and Green’s function depend on both \(\sigma\) and the radius of the ball—expressions for them can be found in the supplementary material.

Monte Carlo estimator. The integral form makes it possible to construct Monte Carlo estimators of \(u(x)\). Starting at any point \(x \in \Omega\), the Walk on Spheres [Muller 1956; Sawhney and Crane 2020] recursively estimates the solution by uniformly sampling a point \(x' \in \partial B(x)\) (see Figure 2a). The source term’s contribution is evaluated by sampling an additional point \(y \in B(x)\) at every step. The position of \(y\) can be chosen proportional to the Green’s function to reduce variance, and Section 5.2 introduces an efficient way to do so.

The radius at each step is arbitrary as long as the resulting ball does not intersect the boundary. It is usually set to the minimum distance to the boundary, as taking the largest admissible sphere
3.2 Walk on Stars

Mixed boundary conditions. A generalization of the earlier family of PDEs additionally considers Neumann boundary conditions:

\[
\Delta u(x) - au(x) = -f(x) \quad x \in \Omega,
\]
\[
u(x) = g(x) \quad x \in \partial \Omega_d,
\]
\[
\partial_n u(x) = h(x) \quad x \in \partial \Omega_n,
\]
where the sets \(\partial \Omega_d\) and \(\partial \Omega_n\) partition \(\partial \Omega\) into regions with Dirichlet and Neumann boundary conditions, and \(h(x)\) specifies the normal derivative \(\partial_n u(x)\) on the boundary. We assume that the Neumann boundary is (locally) convex and smooth. Computing solutions on non-convex Neumann domains is more involved [Sawhney et al. 2023] and beyond the scope of this work.

As before, the problem can be cast into an equivalent recursive integral form. Instead of integrating over a ball, we must now integrate over a star defined as the intersection of the domain and a ball, whose radius equals the minimum distance between \(x\) and \(\partial \Omega_d\) (see Figure 2b). This leads to the following boundary integral equation (BIE):

\[
\kappa(x) u(x) = \int_{\partial S} P^\sigma (x, x') u(x') \, dx' - \int_{\partial S_n} G^\sigma (x, z) h(z) \, dy,
\]
\[
+ \int_{S} G^\sigma (x, y) f(y) \, dy,
\]
\[
\kappa(x) = 1/2 \quad x \in \partial \Omega_1, \quad \kappa(x) = -1 \quad x \notin \partial \Omega_1,
\]
where \(S\) denotes the star, \(\partial S\) is its boundary, and \(\partial S_n = \partial S \cap \partial \Omega_n\) refers to the intersection of the star’s boundary and the Neumann boundary region. If the closest point on the boundary is part of \(\partial \Omega_d\), the expression reduces to Equation (2).

Monte Carlo estimator. Generalizing the previous estimator to Equation 4 leads to the Walk on Stars [Sawhney et al. 2023]. In each iteration, this method samples a random ray starting at \(x\), whose direction is either uniform on the sphere (if \(x \in \Omega\)) or uniform on the hemisphere pointing towards the interior (if \(x \in \partial \Omega\)). Intersecting this ray with \(\partial S\) produces a sample \(x'\) that can be used to estimate the recursive term (first integral). To account for sources (third integral), the method picks a position along the ray proportional to the Green’s function, ignoring samples that fall outside of the star.

To handle Neumann boundaries (second integral), Sawhney et al. [2023] construct a custom bounding volume hierarchy (BVH) to sample the domain \(\partial \Omega_n\). We use simpler scheme that we found to be more suitable for the Bézier spline curve representation used in this article. This method does not require custom data structures in exchange for a potentially larger amount of variance; we explain it in the supplemental material.

Figure 3b illustrates the complete estimator. At each step, we check the type of the closest boundary position to select between the sphere (Dirichlet) and star (Neumann) cases. We then take a suitable step, accumulating contributions according to Equations (2) and (4). This process repeats until reaching a position \(\hat{x}\) within distance \(\epsilon\) to a Dirichlet boundary, where the solution is approximated by \(g(\hat{x})\).

3.3 Spatially-varying coefficients

We can further expand the family of supported PDEs by considering spatial variation of additional terms:

\[
\nabla [\sigma(x) \nabla (u(x))] - \sigma(x) u(x) = -f(x) \quad x \in \Omega,
\]
\[
u(x) = g(x) \quad x \in \partial \Omega_d,
\]
\[
\partial_n u(x) = h(x) \quad x \in \partial \Omega_n,
\]
where \(\sigma(x)\) and \(\alpha(x)\) are spatially-varying diffusion and screening coefficients. In the special case of pure Dirichlet boundaries, Sawhney et al. [2022] show how the combination of a Girsanov transformation and an insight from null-collision random walks in heterogeneous volumes [Woodcock et al. 1965; Galtier et al. 2013] can turn Equation 5 into a screened Poisson equation with constant coefficients (Equation 1):

\[
\Delta \bar{U}(x) - \sigma(x) \nabla (U(x)) = -(\sigma - \sigma'(x)) U(x) - f'(x) \quad x \in \Omega,
\]
\[
U(x) = \sqrt{\sigma(x)} g(x), \quad x \in \partial \Omega_d,
\]
where \(f'(x) = f(x)/\sqrt{\sigma(x)}\), \(\sigma\) is a positive free parameter, and

\[
\sigma'(x) = \frac{\sigma(x)}{\sigma(x)} + \frac{1}{2} \left( \frac{\Delta \sigma(x)}{\sigma(x)} - \frac{\nu^2 \ln(\sigma(x))}{2} \right).
\]

The solution in terms of this modified problem is then given by \(u(x) = \alpha(x)^{-1/2} U(x)\). A similar change is also possible in the more general case of mixed boundary conditions, please see the supplemental material for details. For brevity, we only discuss the Dirichlet version in the following.

Starting from Equation (6), we can build the WoS estimator using Equation (2). Omitting the source term \(f(x)\), we have:

\[
U(x) = \int_{\partial B(x)} P^\sigma (x, x') \, dx' + \int_{B(x)} G^\sigma (x, y) (\sigma - \sigma'(y)) U(y) \, dy.
\]

Compared to the basic screened Poisson estimator, the volume integral over \(B(x)\) now also depends on the solution \(U(y)\). Since both integrals evaluate the solution recursively, Sawhney et al. [2022] probabilistically select one of the two terms in each iteration to prevent exponential runtime complexity. They derive the following
With this background in place, we can now turn to inverse problems. The source term \( Y \) previously introduced estimators solve a Fredholm integral equation. Vector evaluation of the derivative \( \partial \) resulting methods generalize to arbitrary parameter counts. The latter is mainly of interest. For notational simplicity, we focus on a function \( u \) on a regular grid. This solution is assumed to depend on a parameter \( \pi \). Our goal is to solve a minimization problem of the form

\[
\min_{\pi} \ell(u(\pi))
\]

where \( \ell \) is the selection probability using the relation between \( P^\pi \) and \( G^\sigma \):

\[
P^\pi(x, z) = 1 - \sigma |G^\sigma| = 1 - \sigma \int_{B(x)} G^\sigma(x, z) \, dz.
\]

Using this relation, the probability of sampling the volume term is \( P_B = 1 - P_B = 1 - \sigma |G^\sigma| \). The surface and volume terms use the same strategies as the basic screened Poisson estimator. This yields the following estimator:

\[
U(x_k) = \begin{cases} 
U(x_{k+1}) & \text{if } \xi \in [0, 1) \\
U(x_{k+1}) \left( 1 - \frac{\sigma'(x_k)}{\sigma} \right) & \text{otherwise}
\end{cases}
\]

where \( \xi \in [0, 1) \) is a uniform variate. The efficiency of this estimator depends on the choice of the free parameter \( \sigma \). Sawhney et al. [2022] set it to a global constant \( \sigma = \max_{x \in \Omega} \sigma'(x) - \min_{x \in \Omega} \sigma'(x) \). In Section 5.1, we further investigate the choice of \( \sigma \) and propose an acceleration data structure to set it adaptively.

However, even an adaptively chosen \( \sigma \) can produce samples with a very large weight, which adds significant variance especially in long random walks. Sawhney et al. [2022] adopt weight windowing to split the path once its throughput weight exceeds a given threshold, redistributing this weight across independently sampled path suffixes. We will shortly see that this complicates derivative computation, for which we present a solution in Section 4.1.

4 INVERSE SOLVER

With this background in place, we can now turn to inverse problems. Our goal is to solve a minimization problem of the form

\[
\tilde{\pi} = \arg \min_{\pi} \ell(u(\pi))
\]

using gradient-based optimization. The components of the vector \( u(\pi) \) represent discrete evaluations of the PDE’s solution \( u(x) \), e.g., on a regular grid. This solution is assumed to depend on a parameter vector \( \pi \) that controls the source term, boundary values, etc. The function \( \ell \) is a differentiable loss function, for example a \( L_2 \) distance between \( u(\pi) \) and a reference. The optimization depends on our ability to differentiate both \( \ell \) and the solution \( u(\pi) \), of which the latter is mainly of interest. For notational simplicity, we focus on a derivative \( \partial_{\pi} u \) with respect to a single parameter \( \pi \), but the resulting methods generalize to arbitrary parameter counts.

Fredholm integral equations. We begin by observing that all previously introduced estimators solve a Fredholm integral equation of the second kind:

\[
u(x, \pi) = S(x, \pi) + \int_{Y} K(x, x', \pi) u(x', \pi) \, dx'.
\]

This involves a sampling process on a domain \( Y \) involving accumulation of a source term \( S \) and recursive weighting by a kernel \( K \). The source term \( S \) may itself be defined in terms of an integral but does not depend on the solution \( u \). The key challenge we address in the following is the evaluation of the derivative \( \partial_{\pi} u \).

This formulation is reminiscent of the rendering equation [Kajiya 1986] solved by physically-based rendering algorithms:

\[
L_0(x, \omega) = L_c(x, \omega) + \int_{S^2} f_s(x, \omega, \omega') L_i(x, \omega') \, d\omega'.
\]

4.1 Path Replay Backpropagation

PRB is best explained using pseudocode. For this, let us first consider Algorithm 1, which computes a 1-sample estimate of the primal equation (12):

```
1  def solve(x):
2     u = 0  # Running solution estimate
3     \beta = 1  # Path throughput
4     while not done:
5         u += \beta * S(x, \pi)  # Accumulate source term
6         x, new, pdf = sample_K(x)  # Sample integral kernel
7         \mu = K(x, x_new, \pi) / pdf  # Compute sampling weight
8         \beta *= \mu  # Update throughput
9     return u
```

Algorithm 1. Monte Carlo solver for the Fredholm equation (12).

The running estimate \( u \) and throughput \( \beta \) are initially set to 0 and 1. At each step, the iteration accumulates throughput-weighted source contributions (e.g., the source term of the PDE or emission in...
which it does by rewinding the underlying pseudorandom number generator to an earlier state and reproducing the same sequence of random numbers. This phase restarts at position \( x \) and takes a perturbation \( \delta_u \) and the estimate \( u \) of the prior phase as input. It iteratively computes the necessary contribution of the path suffix by subtracting that of the prefix. This involves no approximations and removes the storage overheads.

Algorithm 2 realizes PRB for the previous example (Algorithm 1). The implementation evaluates the \( \pi \)-dependent functions \( S \) and \( K \) using automatic differentiation (AD), where \( \text{backward}() \) backpropagates gradients to \( \delta_\pi \). AD will usually eagerly differentiate the entire program, but here we actually want to localize its usage to a part of the loop body. Arithmetic outside the \text{ad_enabled()} blocks is therefore not tracked to avoid a costly AD computation graph spanning multiple loop iterations. Note that the above pseudocode evaluates certain terms twice (e.g., \( S(x, \pi) \)), which is purely for notational simplicity and can be avoided in practice.

Algorithm 2 realizes a detached sampling strategy [Zeltner et al. 2021; Vicini et al. 2021], which means that it estimates the parameter-dependent integral by only differentiating the integrand \( f(x, \pi) \):

\[
\int_X \frac{\partial f(x, \pi)}{\partial \pi} \, dx \approx \frac{\partial f(x_i, \pi)}{p(x_i, \pi)},
\]

where \( x_i \) is a sample with density \( p(x_i, \pi) \). On the other hand, attached estimators furthermore differentiate the sampling strategy (i.e., how \( x_i \) is generated as a function of \( \pi \)). We use detached estimators throughout the paper but note that attached strategies could be useful to handle derivatives with respect to the domain boundary.

Algorithm 3 presents complete pseudocode of path replay for the WoS estimator of the screened Poisson equation based on the notation introduced in Section 3. The function \( \text{detach}(x) \) disconnects its argument from the automatic differentiation graph. This allows writing Algorithm 2 without duplicating the computation of \( \mu \) once inside and once outside the \text{ad_enabled()} block. The supplemental material provides an extended version of this algorithm that also accounts for Neumann boundary conditions.

### 4.2 Weight windowing

Weight windowing is an area where differences between rendering and random walk–based PDE solvers become apparent. Assuming perfect sampling of \( K \), the weight \( \mu \) in a physically based renderer is upper-bounded\(^1\) by 1—exceeding this value would imply that the material reflects more light than it receives. Non-electromagnetic radiation can sometimes require albedos that violate energy conservation (e.g., neutrons in nuclear reactor simulations), but this case is uncommon in rendering. As a consequence, the throughput \( \beta \) decays over time, enabling efficient stochastic path termination.

Random-walk based PDE solvers that target the spatially variable case do not have such an upper bound. The path weight \( \beta \) can become arbitrarily large, causing estimates with a long-tailed statistical distribution that impedes the optimization. Prior work [Sawhney

---

\( ^1 \)This assumes that the renderer propagates basic radiance, the ratio of radiance and the squared index of refraction.

---

**Fig. 4.** (a) Primal evaluation of a linear path with source \( (S) \) and weight \( (\mu) \) terms. (b) Path Replay Backpropagation (PRB) [Vicini et al. 2021] replays the random walk once more to evaluate the backward derivative of all source and weight terms. The \( \mu \)-derivative depends on the weighted source terms of the path suffix, which the method computes incrementally by subtracting the prefix from the total amount.
et al. 2022] applied weight windows to address this problem, and we adopt this approach here as well.

The idea of a weight window is easily explained: if $\beta$ ever exceeds a certain threshold (say, 2), we duplicate the current state and continue two independent walks starting from this position (Figure 5a). A high weight signals that the path prefix is inadequately sampled, and splitting it increases the computation budget retroactively for this part of the integration domain. We use a simple 2-way split as opposed to a more complex n-way criterion [Sawhney et al. 2022] as we found this to be sufficient to address convergence issues in the spatially variable case. Finally, a standard Russian roulette stochastic stopping criterion is used on the lower end of the window.

A practical issue with branching walks is that their recursive implementation requires an unknown amount of stack memory. Conservative provisioning of a suitably large stack per GPU thread will generally exhaust the device memory, making this strategy impractical. We instead perform the computation in phases: each time, a random walk requires a split, we follow one branch in the current phase and atomically append the other to a global work queue for processing in the next phase. This process repeats until no work is left (Figure 5b). Memory usage then becomes proportional to the actual number of branches per phase, a significant improvement compared to conservative local provisioning of a stack.

2This means that states with $\beta > \beta_{\text{max}}$ will remain outside the window after branching.

Finally, we associate a bit vector with every state that summarizes the branch decisions taken up to that point. The root of a random walk initializes this bit vector with 1, and every subsequent split either appends 0 or 1. The need for this will be clarified in Section 4.3.

4.3 Path replay for branching random walks

The PRB algorithm presented in Section 4.1 efficiently differentiates random walks with a linear structure, but it is unfortunately incompatible with the important weight windowing modification (Section 4.2) needed to handle the spatially variable case. We therefore extend Path Replay to branching random walks. The core mechanism of the method remains unchanged: it performs and subsequently replays random walks to efficiently evaluate the backward derivative. Our adaptation of this scheme to tree structures limits memory usage in exchange for redundant computation.

Section 4.2 explained how our solver processes linear chains in a parallel phase, using subsequent phases to handle splits. The high-level structure of this looks as follows:

```
while len(queue) > 0:
    todo = []
    for orig_state in queue:
        final_state = solve(orig_state, write_branches_to_queue)
        queue = todo
```

PRB would typically be applied to the combination of $(\text{orig}_{\text{state}}, \text{final}_{\text{state}})$ following line 4. However, this strategy will not work for a branching walk when $\text{orig}_{\text{state}}$ refers to a split location from a prior phase. In this case, it is not sufficient to replay from this intermediate state—we must start from the beginning to generate derivative contributions linking prefix and suffix (Figure 5b).

We therefore store the initial configuration (position, pseudo-random number generator state) of the random walk along with the bit vector describing all branching decisions taken to reach the final state. Following this, we can reconstruct all derivative terms for a chain connecting root to leaf, which we additively combine with contributions from other parallel random walks and phases to compute a combined derivative.
The practicality of the redundant prefix traversal relies on the property that splitting is limited to outliers and doesn’t happen excessively often along the same path. A variant on the other extreme of the storage-computation trade-off space could avoid redundant replay by allocating memory at every split to accumulate the weighted contribution of all suffixes originating at this position.

4.4 Validation and results

Validations. We first validate our gradient computations against finite differences. Figure 7 shows the parameter derivatives on a slice through the center of the domain. The coefficients are represented as cubically interpolated 3D textures of resolution $32^3$. Their derivatives are computed by differentiating the $L_2$ norm of a $16^3$ grid of solution evaluations in the domain. Gradients constructed using path replay and weight windowing reveal a perfect match against the reference. We provide similar validations for a 2D problem in the supplementary material.

Optimization results in 2D. Figure 8 shows the evolution of different optimizations that attempt to recover shown reference parameters by measuring the $L_2$ difference between a tentative solution and reference on a $32^2$ grid. The parameters are represented as $16^2$ textures with cubic interpolation. Each experiment optimizes one PDE coefficient, while keeping the others constant. This type of inversion is too ambiguous with just a single observation of the solution, and we generate many different configurations by prescribing linear combinations of sinusoids and linear ramps to parameterize the temperature distribution on the boundary. In each iteration, we select 6 of them at random and use the Adam optimizer to take a step based on the summed gradient. We do not use any regularization but clip tentative parameters values to the range of the reference to improve robustness. In all cases, the difference between reference and tentative solution tends to zero towards the end of the optimization, producing a faithful parameter retrieval. We started all optimizations with a constant texture and did not find the results to be overly sensitive to initial values.

Limitations. The presence of Neumann boundary generally results in longer random walks and higher variance. This increases optimization time due to the need to take more and smaller steps, and reduces the quality of reconstruction results. We analyze this effect by reconstructing the same high-frequency coefficients under Dirichlet and mixed boundary conditions. As shown in Figure 10, the size of the Neumann boundary considerably affects the quality of our reconstructions. To keep the comparison simple, we use the same number of samples across optimizations. This limitation is most pronounced when reconstructing the diffusion coefficients, as high frequency diffusion coefficients cause a high number of path splits. In the limit, large Neumann boundaries can therefore make the inverse solver impractical. We did not implement a 3D solver with Neumann boundaries, but expect similar limitations as in 2D.

We did not compare our method with state-of-the-art inverse FEM solvers, as we currently do not expect MC solvers to outperform these on our simple example problems. FEM solvers have received decades of research and our paper is only a first step in the use of inverse MC solvers. We believe that MC PDE solvers offer valuable flexibility which will lead to further progress on real inverse PDE applications.

Fig. 6. (a) The spatially variable case depends random walks that can branch into separate suffixes. (b) The backward derivative of weights in the prefix preceding the split ($\mu_n$) now involves quantities from both suffixes. Due to linearity, these don’t all have to be computed at the same time. Our method performs replay per phase of the original primal computation (Figure 5b). In doing so, it only accumulates partial derivative terms associated with the subpath connecting roots to leaf nodes processed by the current phase.

Fig. 7. Comparison between our gradient estimate and the finite differences reference (FD) for parameters of the general elliptic PDE (Eq. 5) in 3D.
Fig. 8. Optimization results for different PDE coefficients in a 2D problem. Columns (a): The evolution of the optimized parameter throughout the optimization. Column (b): The reference. Columns (c) and (d): one of the boundary conditions used in the optimization, and the corresponding reference solution inside the domain. Columns (e): The difference between the reference solution in the first and the last iteration of the optimization. The orange boundary lines represent Neumann boundaries whereas dark lines represent Dirichlet boundaries.

Fig. 9. Optimization results for different PDE coefficients in 3D. The result images are structured as in Figure 8. The values of reference and the optimized coefficients are visualized on 2 distinct slices whereas the the solution tensors (the last 3 rows) are visualized on a slice passing through the center of the model.
5 ACCELERATING THE FORWARD SOLVER

One important limitation of the method as presented so far is that the diffusion coefficient must have a sufficiently low frequency. Noisy intermediate steps produced by the gradient-based optimization cause a high majorant \( \sigma \) (Section 3), which in turn slows down the optimizations. We now present an adaptive majorant and sphere radius selection to mitigate this effect. We also propose an optimized Green’s function evaluation routine, that offers further speedups.

Table 2. Average duration of different phases during each optimization step (in seconds). We report timings for computing the primal forward computation (P) and gradient computation (G). In parenthesis, we give the number of samples per point.

|                | 2D | 3D |
|----------------|----|----|
| Source O1      | 0.44 | 0.16 | 0.06 | 0.28 |
| Source O2      | 0.35 | 0.12 | 0.05 | 0.15 |
| Screening O1   | 1.83 | 0.99 | 0.56 | 0.37 |
| Screening O2   | 1.02 | 0.59 | 0.25 | 0.26 |
| Diffusion O1   | 0.73 | 0.87 | 0.70 | 2.98 |
| Diffusion O2   | 0.51 | 0.32 | 0.48 | 1.00 |

5.1 Adaptive majorant and radius selection

We found that the solvers with spatially-varying coefficients can be accelerated by adaptively setting the majorant and picking a sphere radius that is smaller than the maximum admissible one. To explain why and how, we first recall the walk on spheres formulation given in Equation 8:

\[
U(x) = \int P^\sigma(x, x') U(x') \, dx' + \int G^\sigma(x, x') (\sigma - \sigma'(x')) U(x') \, dx'.
\]

Sawhney et al. [2022] globally set \( \sigma \) to \( \max_{x} \sigma'(x) - \min_{x} \sigma'(x) \), which becomes large if the diffusion coefficient \( \alpha(x) \) exhibits local high-frequency variation (see Figure 11).

A high global \( \sigma \) reduces the expected per-step progress of the random walk and increases computation time. In analogy to volume rendering, we characterize the average progress by the mean free path:

\[
r_f(R, \sigma) := \mathbb{E}[r | R, \sigma] = P_{AB} R + P_B \int_0^R \frac{G^\sigma(r)}{|G^\sigma|} \, dr, \tag{16}
\]

where \( \mathbb{E} \) is the expected value operator, \( r = \|x - x'\| \), \( R \) is the radius of the sphere, \( G^\sigma(r) \) is the radial distribution of the Green’s function, \( P_{AB} \) and \( P_B \) are the selection probabilities of sampling the boundary and the volume integral, respectively. This definition is valid for the sampling routines outlined in Section 3.3.

Spatially-varying majorant. Inspired by similar approaches in volume rendering [Yue et al. 2010; Szirmay-Kalos et al. 2011], we increase the mean free path by replacing the global majorant by a spatially-varying one. This prevents local variations from slowing down convergence globally. We use a multi-resolution texture, that in each level stores the maximum value of \( \sigma'(x) \) within a given ball \( B(x, R) \). This is then used as a local majorant \( \tilde{\sigma}(x, R) \). That choice of majorant limits per-step throughput increases to at most a factor of 2 and thus prevents excessive splitting.
Radius selection. We further increase the mean free path by adaptively selecting the radius of the sphere to integrate over. In Figure 12, we plot the Green’s function for varying parameters \( R, \sigma \) and report the corresponding mean free path. For a fixed radius, increasing the majorant reduces \( \overline{P}_{m} \) and \( f^\sigma(r) \) gets peakier close to the origin, hence decreasing the mean free path. For a fixed majorant, increasing the radius initially increases the mean free path. However, the probability of sampling the boundary eventually converges to zero and the shape of the Green’s function also approaches a limit behavior. Empirically, we found that increasing the radius beyond \( R \approx 5/\sqrt{\sigma} \) barely improves the mean free path.

Since \( R \sqrt{\sigma(x, R)} \) is a strictly increasing function in \( R \), we can use bisection to find \( R_0 \) such that \( R_0 \sqrt{\sigma(x, R_0)} \approx 5 \). The value \( R_0 \) is an upper bound, and the optimal radius (in terms of mean free path) is a value between zero and \( R_0 \). We find the optimal value by evaluating \( r_f (R, \sigma(x, R)) \) at equally spaced values of \( R \in [0, R_0] \) using a precomputed lookup table.

Discussion. Figure 13 plots circles generated using our radius selection algorithm. If the center is far from a region with high-frequency variation, the circle is biggest circle excluding that region. For point very close to that region, the radius increases again as in any case the majorant is high. The algorithm still locally needs small steps, but our adaptive radius selection prevents this behavior from expanding to the entire domain. Figure 14 shows histograms of the path length distribution for both the baseline and our method. The average path length in our method is significantly lower in the case of localized high-frequency variation, without any noticeable difference in estimator variance. If the variation is high throughout the domain, there is no significant difference.

5.2 Efficient Sampling of the Green’s Function in 2D

Solving PDEs with WoS and WoSt requires repeated sampling of the associated Green’s function \( G^\sigma(x, y) \). To sample proportional to this function, we generate a random uniform direction, and sample a distance proportional to the radial Green’s function \( G^\sigma(r, z) \). First, we define \( y = r/R \) and \( z = R \sqrt{\sigma} \) to decrease the degrees of freedom in the normalized radial Green’s function \( G^\sigma(r, z)/R \), which we can now write as:

\[
G(y, z) = y \left[ K_0(y z) - I_0(y z) \frac{K_0(z)}{I_0(z)} \right]
\]

where \( I \) and \( K \) refer to modified Bessel functions of the first and second kind (see supplementary material for details). Numerically robust evaluation of Bessel functions is relatively expensive. Given that we use simple boundary representations like quadratic Bézier curves, circles and SDFs, computation of the boundary queries are cheap. We found that evaluation of the Bessel functions is one of the main computational bottlenecks. Thus, we developed a sampling scheme that allowed us to run our simulations 5 to 10 times faster compared to the previous work which uses rejection sampling to sample the Green’s function.

Our goal is to use the inverse transform method to draw \( y \) proportional to \( G(y, z) \) for a given value of \( z \), which requires inversion of the definite integral of this expression.
While this integral can indeed be determined and is given by
\[
I_G(y, z) = \int_0^y G(y', z) \, dy = z^{-2} \left[ 1 - yzI_1(yz) \frac{K_0(z)}{b_0(z)} - yzK_1(yz) \right],
\]
its inversion requires numerical root finding as \(I_G^{-1}\) has no analytic form. This alone would be fine, but the definition (18) in terms of an extended set of Bessel functions makes this even harder than evaluating the original Green’s function (17). We tried various ways of approximating \(I_G(x, y)\) and transformed versions using polynomials but found that large numbers of coefficients were needed to obtain sufficiently good fits.

Near the origin, the modified Bessel function of the second kind resembles the negative logarithm, i.e., \(K_0(z) \approx -\log z\) as \(z \to 0\) [DLMF 2024, 10.30.3], which turns out to be a defining feature that also influences the shape of \(I_G\) elsewhere. We incorporate this observation by adopting an approximation of the form
\[
I_G^a(y, z) = P(y) + Q(y) \log y
\]
where both \(P\) and \(Q\) are polynomials containing only positive even terms, i.e. \(P(y) = p_0y^2 + p_1y^4 + \ldots\) This reduced the absolute error by more than two orders of magnitude compared ordinary polynomials at equal coefficient count. We fit \(I_G\) to \(I_G^a\) by solving a linear system that enforces equality at Chebyshev nodes. The \(p_0\) coefficient encodes the normalization constant. Given that our accelerated sphere tracing method requires to sample the Green’s function in the domain \(y \in [0, 1]\) and \(z \in [0, 5]\), we used 6 coefficients per polynomial which creates a fit with maximum absolute error of \(4 \cdot 10^{-6}\). This error is sufficient for our application as the rest of the solver is implemented in single precision.

To handle different parameters \(y\), we perform many fits and represent the resulting coefficient table as a 1D texture. Importance sampling then performs a texture fetch to obtain coefficients for a given \(y\) followed by numerical inversion using Newton-bisection and the bracketing interval \([0, 1]\). This requires the derivative
\[
\frac{I_G'}{I_G} (x, y) = P'(x) + x^{-1}Q(x) + \log(x)Q'(x),
\]
which is cheap to evaluate in lockstep with \(I_G\) and also provides the needed sampling density. Finally, a good starting guess can greatly reduce the number of Newton-bisection steps. We use
\[
x_0 = 1 - z^{-1} \cosh^{-1}(\cosh(z)(1 - q) + q)
\]
where \(q = \sqrt{\xi}\) for a random variate \(\xi \in U[0, 1]\). In the domain \(y \in [0, 1]\) and \(z \in [0, 5]\), rejection sampling is not very expensive and the main benefit of our method comes from the evaluation of the Green’s function. But Green’s function is extremely peaky for larger \(z\) and rejection sampling should be avoided.

Lastly, we emphasize that the Poisson kernel \(P^a(x, x')\) can be evaluated by \(P^a(x, x') = 1 - |G^\tau|\) where \(|G^\tau|\) is the norm of the Green’s function (see supplementary).

Most of the above steps are somewhat specific to the 2D case due to the complexity of the underlying Green’s function. The 3D case only involves hyperbolic trigonometric functions, so the polynomial approximation is not needed in practice.

6 APPLICATION: ELECTRICAL IMPEDANCE TOMOGRAPHY

6.1 Problem statement

We apply our methods to electrical impedance tomography (EIT), which reveals an object’s internal structure from measured electrical currents. This requires applying and measuring currents at electrodes placed on the object’s surface, and then solving an inverse PDE problem to reconstruct the object’s spatially-varying electrical conductance. Variations in conductance strongly correlate with specific internal structures. Since EIT measurements are on the domain’s boundary and not in the interior, this problem is harder than the somewhat artificial setups studied so far (e.g., in Fig. 8 and 9).

A common application of EIT is non-invasive medical diagnosis, where it can aid detecting breast cancer, blood clots, and pulmonary emboli. Malignant tumors, with their high water content, exhibit conductivity levels around an order of magnitude higher than the surrounding healthy tissue [Adler and Boyle 2017]. Electrical-impedance tomography also finds application geophysics (e.g., to locate mineral deposits) or material sciences (e.g., to identify corrosion and material defects) [Mueller and Siltanen 2012].

PDE formulation. In EIT, the following PDE governs the distribution of voltage \(u(x)\):
\[
\nabla [\alpha(x) \nabla u(x)] = 0, \quad x \in \Omega.
\]
The function \(\alpha(x)\) is the spatially-varying conductance that we seek to reconstruct.

The choice of an electrode model determines the boundary conditions of the above PDE. We use a simple model that represents electrodes as infinitesimal points on the boundary and assumes negligible contact resistance. This results in the following Neumann boundary conditions:
\[
\alpha(x_i) \frac{\partial u}{\partial n}(x_i) = I_i, \quad i = 1, \ldots, n,
\]
where \(x_i\) is the position of electrode \(i\) and \(I_i\) the injected current. At any other point on the domain boundary, the solution’s normal derivative must be zero. The solution of this PDE is defined up to an additive constant. We can compute the missing offset by using that the sum of voltage values at the electrodes needs to be zero:
\[
\sum_{i=0}^n u(x_i) = 0.
\]

Inverse problem. We reconstruct the conductance function by minimizing the difference between simulated voltage values and measured values at the electrodes using gradient descent. For the measurements, we inject positive current \(I\) at one electrode and “receive” the corresponding negative current \(-I\) at another one. Measurements are then taken at all inactive electrodes, since we cannot measure the voltage at an electrode where current is injected. We activate electrodes in skip-\(k\) configurations, where between a pair of active electrodes, \(k\) electrodes are skipped.

6.2 Continuous conductance

Our Monte Carlo gradient estimators are flexible in their representation of the PDE’s parameters. We begin by modeling the unknown
conductance as a continuous function and present a discrete representation later.

Challenges. The forward EIT problem only has Neumann boundaries, and the absence of a screening term implies that no energy dissipates. Monte Carlo PDE solvers terminate either when reaching a Dirichlet boundary, or when, due to a screening term, the throughput decreases sufficiently to apply Russian roulette. Neither of this is the case here, which would lead to infinitely long random walks. This is similar to rendering a room containing only perfectly reflecting mirrors. In EIT, the path contributions become more and more low frequency as the number of interactions increases, eventually converging to a global constant.

We ensure a finite number of interactions by decreasing the throughput by a constant factor α user-defined threshold in each iteration, which results in a small bias in our estimators. Sawhney et al. [2023] achieve a similar effect by introducing a small screening term on walk length after which the throughput is decreased in every iteration, μ: throughput decrease factor. The number of samples per electrode for primal and gradient computation is $2^{15}$, whereas $2^{23}$ to estimate the voltage at each Dirichlet boundary. The conductance function is the optimized conductance in the last iteration of O5 (Figure 15).

Table 3. Bias, variance and computation time due to throughput reduction and added Dirichlet boundaries. $k_d$: number of Dirichlet boundaries, $n_p$: threshold on walk length after which the throughput is decreased in every iteration, μ: throughput decrease factor. The number of samples per electrode for primal and gradient computation is $2^{15}$, whereas $2^{23}$ to estimate the voltage at each Dirichlet boundary. The conductance function is the optimized conductance in the last iteration of O5 (Figure 15).

| $k_d$ | $n_p$ | μ | Bias $\times 10^3$ | Var. $\times 10^4$ | Primal (s) | Gradient (s) |
|-------|-------|---|-------------------|-------------------|------------|-------------|
| 0     | 150   | 0.99 | 2.04             | 25.4              | 0.19       | 4.32        |
| 0     | 50    | 0.90 | 11.8             | 4.34              | 0.05       | 0.27        |
| 3     | ∞     | ~   | 1.03             | 5.34              | 0.10       | 1.17        |
| 3     | 150   | 0.99 | 1.04             | 5.00              | 0.08       | 0.71        |
| 3     | 50    | 0.90 | 14.9             | 2.32              | 0.06       | 0.24        |

ACM Trans. Graph., Vol. 43, No. 6, Article 175. Publication date: December 2024.
Table 4. Average duration of different phases during each optimization step (in seconds). From left to right, we report timings for computing the Dirichlet boundary voltages, primal forward computation and gradient computation. In parentheses, we give the number of samples per Dirichlet boundary (columns 1) or electrode (columns 2, 3).

|                  | Dirichlet boundaries (2^{23}) | Primal (2^{20}) | Gradient (2^{17}) |
|------------------|-------------------------------|----------------|-----------------|
| O1               | 0.29                          | 1.38           | 5.18            |
| O2               | 2.34                          | 1.23           | 3.63            |
| O3               | 1.74                          | 1.35           | 4.58            |
| O4               | 3.65                          | 2.03           | 11.23           |
| O5               | 1.97                          | 2.09           | 12.92           |

For future work, it could be interesting to combine this idea with de-biasing techniques [Misso et al. 2022].

Results. Figure 15 shows reconstruction results using a continuous conductance representation. The boundary conditions are generated with skip-1, skip-3, skip-5 and skip-7 current injection patterns. We compute reference voltage values using our forward solver, but minimize bias by disabling the throughput decrease and only using one Dirichlet boundary inside the domain.

The unknown conductance function $\alpha(x)$ is represented as a bicubically interpolated texture. We use an $L_1$ and a total variation regularizer to encourage sparsity and smoothness, respectively (each at a weight of $10^{-4}$). Close to the boundary, we enforce a constant conductance to ensure that the normal derivative of the conductance is zero on the boundary. As detailed in the supplemental material, this prevents the Neumann boundary condition from turning into a more complex Robin boundary condition. After optimization, the voltage values computed from the optimized coefficients closely match the reference. Due to the ill-conditioned and challenging problem setup, it is no surprise that the reconstructed coefficients are not quite as accurate as for the simpler setups shown previously.

Table 4 shows a breakdown of the computation time of the different phases of our differentiable solver. We use a high number of Monte Carlo samples to compute the fixed voltage values of the circular Dirichlet boundaries. Nevertheless, the main computational bottleneck of our algorithm is the gradient computation, due to its repeated traversal of the branching random walk. This is more pronounced for the last two optimizations, as the higher frequency conductance results in more path splits.

6.3 Discrete conductance

Problem definition. We further apply the differentiable EIT solver to a discrete conductance representation. For this, we split the domain into two disjoint regions, each with a different constant conductance. Our aim is to reconstruct an interior object that is of significantly higher conductance than the surrounding background region. This object can therefore be modeled as a Dirichlet boundary with a constant value$^3$. We illustrate this setup in Figure 16.

Gradient estimator. We estimate gradients for the resulting mixed-boundary problem using differentiable Walk on Stars. Unlike before, we differentiate the original estimator with respect to the shape of the inner Dirichlet boundary $\partial \Omega$. For a point $x$ where the closest boundary is $\partial \Gamma$, we need to differentiate a Walk on Spheres step:

$$\partial_{\nu} u(x, \pi) = \partial_{\nu} \int_{\partial B(x)} \frac{1}{|\partial B(x)|} (\mathbb{1}_{\Omega}(\partial_{\nu}^T(x))u(x, \pi) + \mathbb{1}_{\partial \Gamma}(x)g(x)) \, dx,$$

where $\mathbb{1}_A(x)$ is an indicator function that is 1 if $x \in A$ and 0 otherwise. This formulation makes the termination of walks reaching the epsilon shell $\partial \Gamma$ explicit. The discontinuities due to the indicator function could be handled using a reparameterization [Loubet et al. 2019] or explicit sampling [Li et al. 2018]. However, since $\epsilon$ is small, we found a simpler approach to work well: Whenever a path terminates on a Dirichlet boundary, we compute a solution derivative $\partial_{\tau} u(x, \pi)$ using a first-order expansion along the normal direction:

$$u(x, \pi) \approx V + d(x, \pi) \partial_{\nu} u(x_\pi, \pi),$$  

where $d(x, \pi)$ is the boundary distance, $V$ the constant Dirichlet voltage of the inner object and $x_\pi$ the closest boundary point (see Figure 17a). On the boundary, the spatial solution derivative is continuous and colinear to the normal of the Dirichlet boundary (Figure 17b). We use this local expansion to create the following

$^3$The constant is arbitrary, but can again be pinned after the solve using Equation 23.

ACM Trans. Graph., Vol. 43, No. 6, Article 175. Publication date: December 2024.
Gradient estimator:
\[
\partial_{\pi} u(x_k, \pi) = \begin{cases} 
\partial_{x} V + \partial_{x} d(x_k, \pi) \partial_{\pi} u(x_k, \pi) & x_k \in \partial \Gamma \\
\partial_{x} u(x_{k+1}, \pi) & \text{otherwise}, 
\end{cases}
\]
where \(x_{k,\pi}\) is the projection of \(x_k\) onto the object boundary. We do not differentiate the normal derivative with respect to the parameters and found the bias due to this approximation to be negligible.

Normal derivative. Evaluating the normal derivative very close to the boundary is both noisy and biased due to the proximity to the epsilon shell [Sawhney and Crane 2020]. We therefore evaluate the normal derivative very close to the boundary. As the spatial gradient of the solution \(\nabla u\) is divergence-free in our setup, we correct magnitude of the normal derivative \(\partial_{\pi} u\) by the Jacobian of this transformation (Figure 17c):
\[
\partial_{\pi} u(x_n, \pi) = \left| J(x_n, x_l) \right| \partial_{\pi} u(x_l, \pi),
\]
where \(x_l\) is the offset location, \(J(x_n, x_l) = 1 - \rho(x_n)\) and \(\rho(x_n)\) the curvature at the boundary point. The choice of \(t\) exposes a bias-variance tradeoff. Figure 18 illustrates the effect of \(t\) on the estimated normal derivative.

Gradient validation. We validate our gradients against finite differences (FD) computed at different offsets. (d) Applying the Jacobian correction factor reduces the bias due to an increase in offset distance.

Fig. 18. Illustration of the bias due to evaluating the normal derivative at an offset. (a) The used example setup, where red and purple points correspond to the vertical lines in other plots (in clockwise order), and the green points are the electrodes. The radius of the outer boundary (yellow) is 1. (b) Curvature at the boundary. (c) The normal derivative of the solution computed at different offsets. (d) Applying the Jacobian correction factor reduces the bias due to an increase in offset distance.

Fig. 19. Validation of our derivative estimator against finite differences (FD). We compute derivatives of the solution at the electrodes with respect to vertical and horizontal translations of the inner shape. Starting with 0 at the top, electrodes are numbered in clockwise order. Injected current locations are marked in red (electrodes 0, 10) and the radius of the boundary is 1.

Optimization results. We show optimization results for synthetic example problems in Figure 20. The discrete representation reduces ambiguities compared to the continuous case. We noticed that concave regions are more difficult to reconstruct (e.g., result O4), since the local voltage values become very similar to the voltage of the Dirichlet boundary.

Finally, we apply our method to measured data of metal conductors in a saline solution, using the dataset by Hauptmann et al. [2018]. The dataset is created using the KIT4 (Kuopio Impedance Tomography) measurement system [Hauptmann et al. 2017]. The setup is vertically uniform so that it can be modeled on a 2D domain. We show reconstructions using the SDF representation in Figure 21 and, as an alternative, using a parameterized circle in Figure 1. Both SDF and circle parameterization allows reconstructing the shape and position of the metal conductor. The reconstructions are not perfect, which is likely due to the approximations taken by the electrode model. Further improvements could be made by using more realistic electrode models or formulating the problem with Robin boundary conditions [Maire and Simon 2015; Miller et al. 2024].

7 IMPLEMENTATION DETAILS

We now briefly discuss some key implementation details.

Evaluation of boundary conditions. For all optimization results, we compute gradients using multiple different boundary conditions. Since the boundary values do not affect sampling, we improve efficiency by simultaneously computing the solution for six different boundary conditions (instead of running independent solves).

Circular Dirichlet boundaries. In each optimization iteration for EIT, we find the circular Dirichlet boundaries by using K-means clustering on the value and norm of the conductance function. We eliminate clusters that are too close to each other and pick the top three clusters according to the average conductance over their respective region. We then estimate the voltage at each selected location and use the result during gradient computation. This choice
of Dirichlet boundary locations is physically-motivated: The Dirichlet boundaries can be understood as regions of infinite conductance. Therefore, placing them in high conductance regions reduces bias.

Normal derivatives. The shape derivatives in discrete EIT require estimating the solution’s normal derivative. Even when evaluated at an offset way from the boundary, the normal derivative is noisy. We thus precompute it for each texel of the SDF using $\approx 2^{14}$ samples. For the circle representation (Figure 1), we precompute the normal derivative for an evenly distributed set of points on the circle itself.

Handling interpolated SDFs. The distance values in interpolated SDFs can be inaccurate, especially at low resolutions (see Figure 22). Our discrete EIT gradients rely on an accurate, differentiable distance evaluation. We therefore correct for inaccuracies by dividing the distance by the norm of the spatial gradient of the SDF when evaluating Equation 25. Similar issues with interpolated SDFs were observed in differentiable rendering [Vicini et al. 2022].

Redistancing. To ensure that a texture represents a valid signed distance function, we apply redistancing after each optimization step.
We do this using the scikit-fmm [Furtney 2023] library. It does not account for the bicubic interpolation that we use during simulation. To compensate for this, we upsample the SDF to a resolution of $1024^3$ using bicubic interpolation, then perform the redistancing and finally downsample back to the original resolution.

8 CONCLUSION

We introduced practical derivative estimators for Walk on Spheres and various extensions thereof. Differentiable Monte Carlo methods admit a great flexibility in the underlying representation and set of optimized parameters. For practical applications, this could allow to not only reconstruct the main parameters of interest, but also simultaneously calibrate measurement setups. In medical EIT, determining electrode locations or body shape of patients is still an active research area. Unlike MC solvers, FEM-based methods would have to run a costly domain remeshing step in each iteration. Another benefit of MC PDE solvers is their support for a wide range of parameter representations. The shape of the boundary, boundary conditions, or PDE coefficients can be parameterized by various functions, enabling the use of complex, domain-specific priors. We demonstrate this for EIT, where we use circles and SDFs for discrete shape representations.

The main limitation of our method is the performance and generalizability of the forward solvers. For example, high-frequency spatial variation in the diffusion coefficient requires a large number of costly path splits. These problems are exacerbated for pure Neumann problems without screening term. The effect of parameters on the final PDE solution is often the result of very long random walks, which means that, unlike in rendering, we cannot reduce the number of interactions to a fixed maximum.

We believe that there are many interesting future directions to explore. Our application to EIT only scratches the surface of potential practical usecases. We expect novel applications to come with their own set of challenges, but hope that our methods provide a useful foundation. The similarities with light transport simulation and inverse rendering mean that their solution can benefit from the computer graphics community.

9 ACKNOWLEDGEMENTS

We thank Rohan Sawhney and Dario Seyb for their valuable clarifications on the forward solvers.
