Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-Based Methods on Volumetric Domains

ROHAN SAWHNEY and KEENAN CRANE, Carnegie Mellon University

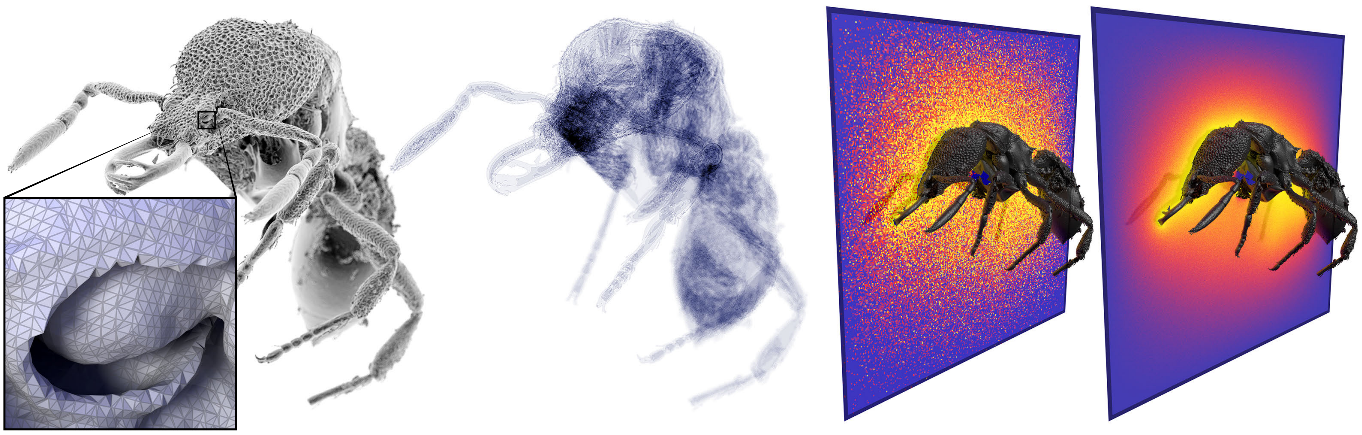


Fig. 1. Real-world geometry has not only rich surface detail *(left)* but also intricate internal structure *(center)*. On such domains, FEM-based geometric algorithms struggle to mesh, setup, and solve PDEs—in this case taking more than 14 hours and 30GB of memory just for a basic Poisson equation. Our Monte Carlo solver uses about 1GB of memory and takes less than a minute to provide a preview *(center right)* that can then be progressively refined *(far right)*. *[Boundary mesh of Fijian strumigenys FJ13 used courtesy of the Economo Lab at OIST.]*

This paper explores how core problems in PDE-based geometry processing can be efficiently and reliably solved via grid-free Monte Carlo methods. Modern geometric algorithms often need to solve Poisson-like equations on geometrically intricate domains. Conventional methods most often mesh the domain, which is both challenging and expensive for geometry with fine details or imperfections (holes, self-intersections, *etc.*). In contrast, grid-free Monte Carlo methods avoid mesh generation entirely, and instead just evaluate closest point queries. They hence do not discretize space, time, nor even function spaces, and provide the exact solution (in expectation) even on extremely challenging models. More broadly, they share many benefits with Monte Carlo methods from photorealistic rendering: excellent scaling, trivial parallel implementation, view-dependent evaluation, and the ability to work with any kind of geometry (including implicit or procedural descriptions). We develop a complete "black box" solver that encompasses integration, variance reduction, and visualization, and explore how it can be used for various geometry processing tasks. In particular, we consider several fundamental linear elliptic PDEs with constant coefficients on solid regions of $\mathbb{R}^n$. Overall we find that Monte Carlo methods significantly broaden the horizons of geometry processing, since they easily handle problems of size and complexity that are essentially hopeless for conventional methods.

Authors' address: Rohan Sawhney; Keenan Crane, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
0730-0301/2020/7-ART1 $15.00
https://doi.org/XX

CCS Concepts: • **Computing methodologies → Shape analysis**.

ACM Reference Format:
Rohan Sawhney and Keenan Crane. 2020. Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-Based Methods on Volumetric Domains. *ACM Trans. Graph.* 38, 4, Article 1 (July 2020), 18 pages. https://doi.org/XX

1 INTRODUCTION

The complexity of geometric models has increased dramatically in recent years, but is still far from matching the complexity found in nature—consider, for instance, detailed microstructures that give rise to physical or biological behavior (Fig. 1). PDE-based methods provide powerful tools for processing and analyzing such data, but have not yet reached a point where algorithms "just work": even basic tasks still entail careful preprocessing or parameter tuning, and robust algorithms can exhibit poor scaling in time or memory. Monte Carlo methods provide new opportunities for geometry processing, making a sharp break with traditional finite element methods (FEM). In particular, by avoiding the daunting challenge of *mesh generation* they offer a framework that is highly scalable, parallelizable, and numerically robust, and significantly expands the kind of geometry that can be used in PDE-based algorithms.

Photorealistic rendering experienced an analogous development around the 1990s: finite element radiosity [Goral et al. 1984] gave way to Monte Carlo integration of the light transport equation [Kajiya 1986], for reasons that are nicely summarized by Wann Jensen [2001, Chapter 1]. Although this shift was motivated in part by a desire for more complex illumination, it has also made it possible to work with scenes of extreme *geometric* complexity—modern renderers handle trillions of effective polygons [Georgiev et al. 2018]

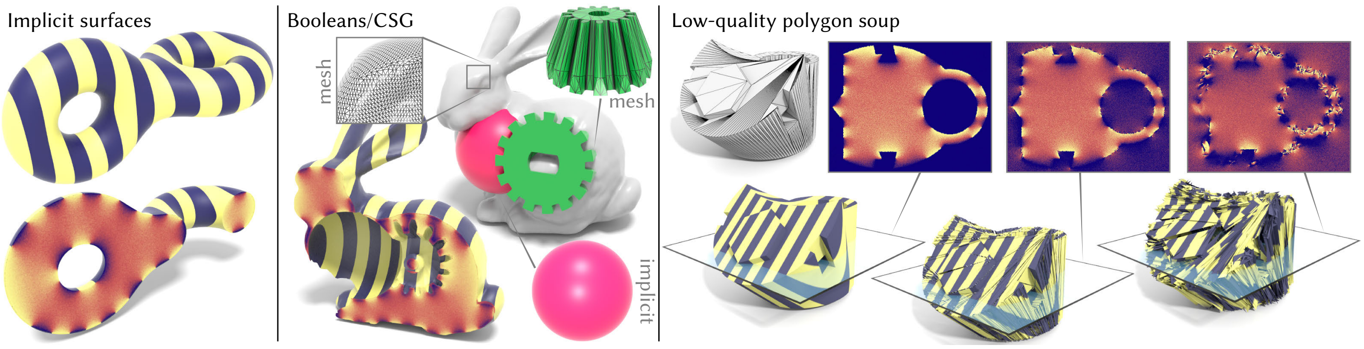


Fig. 2. We can directly solve PDEs on boundary representations not handled by conventional solvers. *Left:* discontinuous boundary conditions on an implicit surface are captured exactly, whereas FEM or BEM would need a fine mesh to approximate such features. *Center:* we avoid the daunting challenge of explicit *mesh booleans*, and can in fact combine heterogeneous representations via constructive solid geometry (CSG). *Right:* as in rendering, we can compute meaningful solutions for geometry with extremely poor element quality, which degrade gracefully even in the presence of significant noise.

and, in stark contrast to FEM, yield high-quality results even for low-quality polygon soup. Moreover, Monte Carlo rendering can use rich geometric representations beyond polygon meshes, since all geometric evaluation boils down to simple ray intersection queries.

The methods described in this paper provide an analogous approach to geometry processing: ray intersection tests are replaced by closest point queries, and recursive ray tracing is replaced by the recursive *walk on spheres (WoS)* algorithm of Muller [1956], as outlined in Sec. 2.2.1. This approach has a number of benefits:

- **Geometric Flexibility.** It works directly with polygonal meshes, NURBS/subdivision surfaces, implicit surfaces, constructive solid geometry, *etc.* Procedural (*e.g.*, instanced) geometry can be used without consuming significant memory.
- **Geometric Robustness.** Geometry need not be watertight, manifold, orientable, nor free of self-intersections; no preprocessing or tessellation/discretization is needed. Sharp edges, small details, and thin features are exactly preserved.
- **Scalability.** The main cost is a bounding volume hierarchy (BVH) for closest point queries, which exhibits $O(n \log n)$ time and memory complexity with respect to the size of the *boundary* geometry, rather than an interior (volumetric) mesh.
- **Parallelism.** It is trivial to achieve near-perfect parallel scaling, and many operations are easily vectorized.
- **Correctness.** Since there is no discretization of space, time, nor function spaces, one obtains the exact solution *in expectation*, *i.e.*, error is due purely to variance in the Monte Carlo estimator, and can be reduced by simply taking more samples.
- **Adaptivity.** Adaptive sampling akin to *radiance caching* [Ward et al. 1988] significantly reduces cost in smooth regions; progressive sampling enables rapid previews of PDE solutions.
- **Output Sensitivity.** The solution can be evaluated at points or regions of interest (*e.g.*, a small window or a slice plane), without having to first perform a global solve.
- **Compatibility.** Monte Carlo methods fit easily into the standard geometry processing pipeline, since they can be used as "black box" solvers that return reliable and accurate solution values at any given query point (*e.g.*, at mesh vertices).

Monte Carlo methods are not, however, a silver bullet. On simple domains with smooth boundary conditions FEM is quite mature and hard to beat in terms of solve time; here Monte Carlo methods can be slow to eliminate high-frequency noise (though see Sec. 5.2.4). For more complex problems, however, end-to-end performance depends on many factors beyond the core solve: mesh generation, parallel scaling, visualization, *etc.* (see Sec. 7 for a more in-depth discussion). It may also not be obvious how to formulate a Monte Carlo estimator for a given PDE. For instance, WoS is not suited for PDEs with variable coefficients, and we do not here consider Neumann boundary conditions (needed for, *e.g.*, linear elasticity). However, WoS can be generalized far beyond the basic PDEs considered in this paper, to include both linear and nonlinear elliptic, parabolic, and hyperbolic equations [Bossy et al. 2015; Shakenov 2014; Pardoux and Tang 1999]. Moreover, Monte Carlo methods for PDEs are far broader than just WoS [Higham 2001; Kloeden and Platen 2013], and the basic framework presented here can be enhanced significantly by drawing on deep knowledge from fields like stochastic control, mathematical finance, and Monte Carlo rendering (Sec. 8). On the whole, Monte Carlo methods have some unique features that make them well-suited to geometry processing, and fill a place in the landscape that has not yet been well-explored.

1.1 Contributions

Our approach builds on grid-free Monte Carlo methods for PDEs, which use closed-form distributions to exactly model large steps of a *continuous* random process like Brownian motion. This type of method has been used for specific problems in, *e.g.*, molecular dynamics [Mascagni and Simonov 2004], integrated circuit design [Coz and Iverson 1992], porous media [Hwang et al. 2000], and electrostatics [Hwang and Mascagni 2004]. However, no general framework has been developed for the types of problems arising in geometry processing, since traditional applications of such methods need not consider, *e.g.*, defective or highly detailed geometry, derivative estimation, or visualization of solutions. This paper develops a holistic Monte Carlo framework for geometry processing based on linear elliptic PDEs with constant coefficients on a volumetric domain, *i.e.*, an n-dimensional solid region in $\mathbb{R}^n$. In particular, we provide:

– a unified discussion of estimators for PDEs commonly used in geometry processing algorithms (Sec. 2),
– strategies for estimating derivatives and standard differential operators (Sec. 3),
– variance reduction strategies for PDE and derivative estimators (Sec. 4),
– geometric representations that circumvent common meshing / preprocessing challenges (Sec. 5.1),
– visualization techniques for scalar- and vector-valued data that significantly enhance performance (Sec. 5.2), and
– strategies for implementing core geometry processing algorithms within the Monte Carlo framework (Sec. 6).

Our approach also connects to a large body of work on Monte Carlo rendering [Dutre et al. 2006; Pharr et al. 2016]. From the PDE point of view, the major difference is that the differential equation governing radiative transfer is first order in space, whereas the PDEs we seek to solve have second order, diffusive terms that demand different numerical techniques. There are of course many parallels between these problems from a mathematical, computational, and system design point of view, which we explore throughout the paper.

2 PDE ESTIMATORS

We begin with estimators for some of the most fundamental PDEs in geometry processing—in general, there are two complementary points of view. One, based on *potential theory*, is to express the solution as a recursive integral equation (akin to the rendering equation of Kajiya [1986]), and apply Monte Carlo integration. The other, based on *stochastic calculus*, is to express the solution in terms of a random process, and use Monte Carlo to simulate random walks. This duality is well-illustrated by the WoS algorithm for the Laplace equation (Sec. 2.2), which is our starting point for all other estimators. Whereas the potential theory viewpoint is often simpler to understand, stochastic calculus provides sophisticated tools to analyze and relate random processes to the integral formulation of PDEs they model–we primarily consider the former in this paper, though use results from the latter where necessary.

2.1 Background and Notation

2.1.1 Notation. For any region $A \subset \mathbb{R}^n$, we use ∂A to denote its boundary, $|A|$ to denote its volume, and $\mathcal{U}(A) = 1/|A|$ to denote the uniform probability density function on A. Throughout we use $\Omega \subset \mathbb{R}^n$ to denote the domain of interest, and $B(x)$ to denote a ball contained in Ω and centered around a point x. For a point $x \in \Omega$, we use $\overline{x}$ to denote the closest point on $\partial\Omega$. Finally, we use Δ to denote the negative-semidefinite Laplace operator on $\mathbb{R}^n$, and $X \cdot Y$ to denote the standard dot product of vectors $X, Y \in \mathbb{R}^n$.

2.1.2 Monte Carlo Integration. To keep exposition self-contained we recall some elementary facts about Monte Carlo integration; see Pharr et al. [2016, Chapter 13] for a more thorough introduction. The basic idea is that an integral can be estimated by sampling the integrand at randomly-chosen points. More precisely, let f be an (L^1) integrable function on a domain Ω. Then the integral

$$I := \int_\Omega f(x)\, dx$$

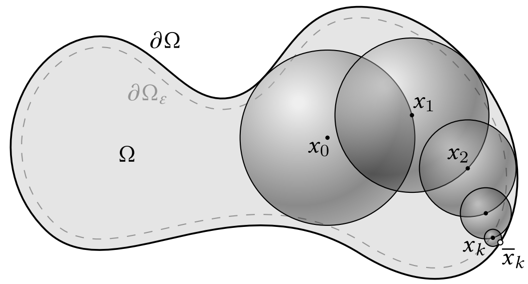


Fig. 3. The walk on spheres algorithm repeatedly jumps to a random point on the largest sphere centered at the current point x_i, until it gets within ε of the boundary. Since the largest sphere can be determined from a simple closest point query, no spatial discretization is needed.

is equal to the expected value of the *Monte Carlo estimator*

$$F_N \;:=\; |\Omega| \frac{1}{N} \sum_{i=1}^{N} f(X_i), \quad X_i \sim \mathcal{U}(\Omega) \tag{1}$$

where N is any positive integer, and $X_i \sim \mathcal{U}(\Omega)$ indicates that X_i are independent random samples drawn from the uniform distribution on Ω. The error of F_N is characterized by its *variance*: its expected (squared) deviation from the true value I.

Importance Sampling. More generally, let p be any probability distribution on Ω that is nonzero on the support of f. Then the integral of f is equal to the expected value of the estimator

$$\frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}, \quad X_i \sim p. \tag{2}$$

Typically, p is chosen to reduce the variance of the estimate by focusing on "important" features in the integrand [Pharr et al. 2016, Section 13.10]. For simplicity, our initial discussion considers only the basic Monte Carlo estimator (Eqn. 1); importance sampling strategies are discussed in Sec. 4.2.

2.1.3 PDE Estimation. The solution to a linear elliptic PDE can be expressed as a linear combination of contributions from the boundary term and the source term. Estimation of these two terms is well-illustrated via the Laplace equation (Sec. 2.2) and Poisson equation (Sec. 2.3), *resp.* One can then build on these estimators to solve other common equations such as the screened Poisson (Sec. 2.4) and biharmonic equations (Sec. 2.5).

2.2 Laplace Equation

Laplace equations are commonly used to interpolate given boundary data $g : \partial\Omega \to \mathbb{R}$ (encoding, *e.g.*, color or deformation) over the interior of the domain. The solution u satisfies the PDE

$$\begin{aligned} \Delta u &= 0 \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega. \end{aligned} \tag{3}$$

These so-called *harmonic functions* have two important characterizations, which are illustrated in Fig. 4:

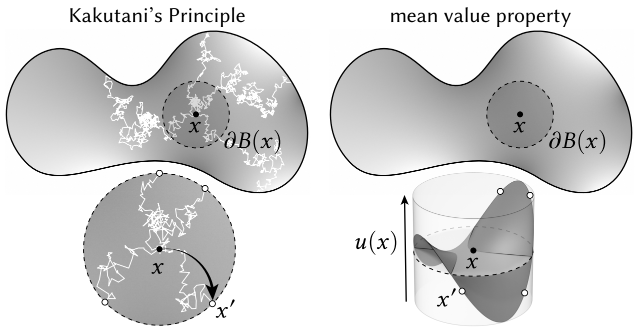


Fig. 4. At any point x, the solution u to a Laplace problem is equal to the average boundary value reached by random walkers *(top left)*, and the average value of u over a sphere $\partial B(x)$ around x *(top right)*. *Bottom:* both quantities can be estimated by recursively sampling points x' on a sphere.

- **Kakutani's Principle** [Kakutani 1944]. The solution value $u(x)$ at any point $x \in \Omega$ is equal to the expected value $\mathrm{E}[g(y)]$, where $y \in \partial\Omega$ is the first boundary point reached by a random walk starting at x (more formally, a *Wiener process*).
- **Mean Value Property** [Axler et al. 2013, Chapter 1]. The value of u at x equals the mean value of u over the boundary of any ball $B(x) \subset \Omega$:

$$u(x) = \frac{1}{|\partial B(x)|} \int_{\partial B(x)} u(y)\, dy. \tag{4}$$

2.2.1 Walk on Spheres Algorithm. Both characterizations suggest algorithms for computing harmonic functions. One idea is to take random steps on a grid, or finite steps in random directions, but these strategies introduce spatial and temporal discretization (*resp.*). An ingenious idea proposed by Muller [1956] is to instead take a *walk on spheres (WoS)*. By symmetry, a continuous random walk starting at a point x_0 is equally likely to exit any point on a sphere around x_0—independent of how long the walk takes, or where it goes while inside the sphere. Hence, we get a statistically perfect simulation by repeatedly jumping to a random point x_{k+1} uniformly sampled from a sphere around the current point x_k (Fig. 3). Once x_k is within a small distance $\varepsilon > 0$ of the domain boundary, we grab the boundary value $g(\overline{x}_k)$ at the closest point $\overline{x}_k \in \partial\Omega$. Apart from a negligible bias introduced by the ε-shell $\partial\Omega_\varepsilon$ (Sec. 6.1), the expected value of this estimator is the exact value of $u(x_0)$, and variance can be reduced by averaging the result of many walks.

The WoS algorithm can also be motivated by the mean value property. At x_0, we can approximate Eqn. 4 via one-point Monte Carlo quadrature, *i.e.*, by picking a random point x_1 on $\partial B(x)$ and evaluating $u(x_1)$. Since $u(x_1)$ is also unknown, we recursively apply this procedure until we reach $\partial\Omega_\varepsilon$, and again grab the closest boundary value $g(\overline{x}_k)$. The analogy from rendering is basic path tracing, where a single Monte Carlo sample is used to recursively estimate the incoming radiance until we hit a light source.

The final Monte Carlo estimate for $u(x_0)$ is given by $\frac{1}{N}\sum_{i=1}^{N} \widehat{u}_0(x_0)$, where $\widehat{u}_0$ is the WoS estimator for the Laplace equation:

$$\widehat{u}_0(x_k) := \begin{cases} g(\overline{x}_k), & x_k \in \partial\Omega_\epsilon \\ \widehat{u}_0(x_{k+1}), & \text{otherwise.} \end{cases} \tag{5}$$

To achieve fast convergence, it usually makes sense to draw x_{k+1} from a uniform distribution on the *largest* sphere around x_k, which can be efficiently computed via closest point queries (Sec. 5.1). In this case one can show that the number of steps needed to reach the ε-shell $\partial\Omega_\varepsilon$ is typically $O(\log 1/\varepsilon)$ [Binder and Braverman 2012].

2.3 Poisson Equation

The Poisson equation, used for tasks ranging from surface reconstruction [Kazhdan et al. 2006] to shape editing [Yu et al. 2004], introduces a *source term* $f : \Omega \to \mathbb{R}$:

$$\begin{aligned} \Delta u &= f \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega. \end{aligned} \tag{6}$$

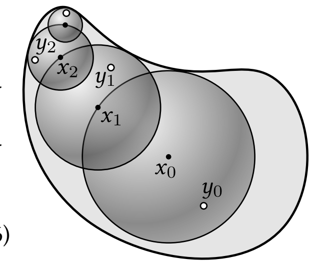


Fig. 5. A source term f can be approximated via a single random sample y_i at each step.

The solution value at any point $x \in \Omega$ can be expressed via another recursive integral equation [Delaurentis and Romero 1990] which generalizes the mean value property:

$$u(x) = \frac{1}{|\partial B(x)|} \int_{\partial B(x)} u(y)dy + \int_{B(x)} f(y)G(x,y)dy. \tag{7}$$

Here G is the *harmonic Green's function* on the ball $B(x)$ (and *not* the Green's function on $\mathbb{R}^n$); see App. B.1. For each step of the walk, we must now estimate the contribution of the source term. A simple strategy is to just use a one-point Monte Carlo estimate [Elepov and Mikhailov 1969], leading to a WoS estimator

$$\widehat{u}_f(x_k) := \begin{cases} g(\overline{x}_k), & x_k \in \partial\Omega_\epsilon \\ \widehat{u}_f(x_{k+1}) + |B(x_k)| f(y_k) G(x_k, y_k), & \text{otherwise.} \end{cases} \tag{8}$$

At each step the sample point y_k is drawn from the distribution $\mathcal{U}(B(x_k))$ on the largest solid ball around x_k (Fig. 5). A proof that this one-point estimator converges to the true value of $u(x)$ can be found in [Delaurentis and Romero 1990, Section II]; see also Fig. 15.

2.4 Screened Poisson Equation

Another common equation in geometry processing (used for, *e.g.*, surface filtering [Chuang and Kazhdan 2011] and computing geodesic distance [Crane et al. 2017]) is the *screened Poisson equation*

$$\begin{aligned} \Delta u - cu &= f \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega, \end{aligned} \tag{9}$$

where $c > 0$ is a constant. The WoS estimator for this equation is identical to Eqn. 8, except that G is replaced by the *Yukawa potential* G_c (App. B.2), and the value of $\widehat{u}_f(x_{k+1})$ is weighted by a factor C that depends on dimension (App. B.2.1).

2.5 Biharmonic Equation

Higher-order PDEs can be estimated by "nesting" lower-order estimators. For instance, the *biharmonic equation*, used in geometry processing for, *e.g.*, deformation [Jacobson et al. 2011] and shape correspondence [Lipman et al. 2010], is given by the 4th-order PDE

$$\begin{aligned} \Delta^2 u &= 0 \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega, \\ \Delta u &= h \quad \text{on } \partial\Omega. \end{aligned} \tag{10}$$

To solve this equation via Monte Carlo, we can re-write it as a system of two 2nd-order PDEs via the substitution $v := \Delta u$:

$$\begin{aligned} \Delta u &= v \quad \text{on } \Omega, & \Delta v &= 0 \quad \text{on } \Omega, \\ u &= g \quad \text{on } \partial\Omega, & v &= h \quad \text{on } \partial\Omega. \end{aligned} \tag{11}$$

To evaluate u at a point x_0 we can apply the standard Poisson estimator (Eqn. 8), using the Laplace estimator (Eqn. 5) to estimate $v(x_k)$ at each step of the walk. However, this naïve strategy is quite expensive since we now have to simulate a whole walk inside each step—Sec. 4.3 describes a more efficient approach.

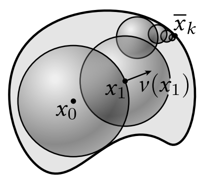
2.6 Exterior Problems

A nice feature of Monte Carlo is that estimators can easily be used to solve PDEs on the domain *exterior*, *i.e.*, the complement of Ω in $\mathbb{R}^n$. To do so, we just apply standard *Russian roulette* [Pharr et al. 2016, Section 13.7] to terminate walks that wander far from the domain boundary. Examples are shown in Fig. 2.

3 DERIVATIVE ESTIMATORS

Geometry processing often requires not only the solution to a PDE, but also derivatives of that solution. Surprisingly little has been said about estimating derivatives via WoS—to our knowledge, the only such work is Elepov and Mikhailov [1969], which briefly discusses the gradient but ignores other differential operators, higher-order derivatives, and variance reduction strategies (Secs. 4.1.2 and 4.1.3). We provide basic derivative estimators derived via potential theory and the more general framework of *Malliavin calculus* [Bell 2012].

3.1 Gradient



As shown in App. A, the gradient of the solution u to a Laplace equation (Eqn. 3) with respect to the evaluation point x can be expressed via a mean value-like principle

$$\nabla u(x) = \frac{1}{|B(x)|} \int_{\partial B(x)} u(y)\nu(y)\, dy, \tag{12}$$

where for a ball of radius R, $\nu(y) := (y - x)/R$ is the outward unit normal at y. A basic WoS estimator for ∇u is then

$$\widehat{\nabla u_0}(x_0) := \frac{n}{R}\widehat{u}_0(x_1)\nu(x_1), \quad x_1 \sim \mathcal{U}(\partial B(x_0)), \tag{13}$$

i.e., just sample a point x_1 on a ball around x_0, and multiply the normal by the WoS estimate for $u(x_1)$ (Eqn. 5). The coefficient n/R comes from the surface area to volume ratio for an n-dimensional ball. Notice that estimating the gradient adds virtually no cost on top of estimating the solution value at x_0—just multiplying the initial normal $\nu(x_1)$ by the final boundary value $g(\overline{x}_k)$.

In the case where we also have a nonzero source term f (Eqn. 6), the gradient can be expressed as

$$\frac{1}{|B(x)|} \int_{\partial B(x)} u(y)\nu(y)\, dy + \int_{B(x)} f(y)\nabla G(x,y)\, dy \tag{14}$$

The WoS estimator just adds a single sample of the latter integral for each step of the walk. This same estimator can be used directly for a biharmonic equation (which is expressed in terms of a Poisson equation); for a screened Poisson equation one simply replaces ∇G with ∇G_c (App. B.2).

3.2 First-Order Differential Operators

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

$$D_Z u(x) = Z \cdot \nabla u(x).$$

All other first-order differential operators, such as divergence or curl, can then be expressed via the partial derivatives $\partial u/\partial e_i = D_{e_i} u$ along the coordinate directions $e_1, \ldots, e_n$ (see Sec. 6 for examples).

3.3 Higher Order Derivatives

If u is the solution to a Poisson equation (Eqn. 6), then its Hessian $\nabla^2 u$ can be expressed via the integral formula

$$\begin{aligned} \nabla^2 u(x) \;=\; & \tfrac{n^2}{R^4} \tfrac{1}{|\partial B(x)|} \int_{\partial B(x)} g(y)(y-x)(y-x)^\top\, dy \\ & - \tfrac{n}{R^2} \tfrac{1}{|\partial B(x)|} \int_{\partial B(x)} g(y) I\, dy + \int_{B(x)} f(y)\nabla^2 G(x,y)\, dy, \end{aligned}$$

where R is the radius of $B(x)$, and $I \in \mathbb{R}^{n \times n}$ is the identity matrix. The WoS estimator for $\nabla^2 u(x_0)$ could in principle then be written as

$$\widehat{u}_f(x_1)\left(\frac{n^2}{R^4}(x_1 - x_0)(x_1 - x_0)^\top - \frac{n}{R^2} I\right) + |B(x_0)| f(y_0)\nabla^2 G(x_0, y_0), \tag{15}$$

where $x_1 \sim \mathcal{U}(\partial B(x_0))$ and $y_0 \sim \mathcal{U}(B(x_0))$. Here we run into some difficulty if the Hessian of the Green's function $\nabla^2 G$ involves terms that behave like Dirac deltas (which will happen, *e.g.*, for the harmonic Green's function): without an importance sampling strategy akin to those used for point sources (Sec. 4.2.2), important contributions will be missed. However, we can apply integration by parts to obtain a different expression for this term, namely

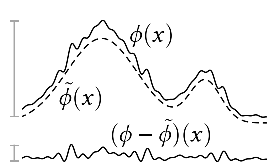
$$\int_{B(x)} \nabla_y f(y) \nabla_x G(x,y) - f(y)(\psi(x,y)\,(y-x)(y-x)^\top - \phi(x,y)\, I)\, dy,$$

where ∇_y denotes the gradient with respect to y, and ψ and ϕ are functions that depend on the particular choice of Green's function (see App. B.2.2). This quantity can then be estimated via, *e.g.*, a single Monte Carlo sample. As with the gradient, other 2nd-order differential operators can then be estimated via the Hessian estimate. See Fig. 6 for a numerical example, and Fig. 15 for convergence plots.

4 VARIANCE REDUCTION

To date, there has been little work on applying variance reduction to WoS—we here give several strategies that improve on the basic PDE and derivative estimators from Secs. 2 and 3.

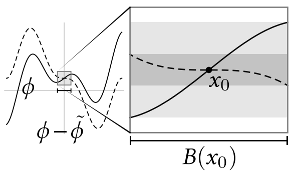
4.1 Control Variates



Suppose we want to estimate an integral $\int_\Omega \phi(x)\, dx$, and have a function $\tilde{\phi}(x)$ with known integral $c \in \mathbb{R}$. A *control variate* strategy is then

$$c + \frac{1}{N}\sum_{i=1}^{N} \phi(X_i) - \tilde{\phi}(X_i),$$

i.e., estimate the *difference* between ϕ and $\tilde{\phi}$, then shift by c. Intuitively, if $\tilde{\phi}$ is similar to ϕ, then the difference will have smaller variance than ϕ itself (see [Veach 1997, 2.5.3] for further discussion).



Control Variates for PDEs. Suppose we let $\tilde{\phi}$ be a low-order Taylor approximation of ϕ around some point x_0. Then in a small neighborhood of x_0 the function $\phi - \tilde{\phi}$ will look "flat," *i.e.*, it will have low variance. This control variate is useful for WoS estimators, which seek to integrate a function over a (typically small) sphere or ball. Though we do not know the terms of the Taylor series *a priori*, we can use running derivative estimates to get an increasingly good guess. In fact, when both solution and gradient strategies are used in conjunction, they reinforce each-other: the variance of the solution estimator is reduced by the gradient estimate, and vice versa. In practice, we hence use these two strategies for all PDEs.

4.1.1 Control Variate for Solution. For a PDE with solution u, let $\overline{\nabla u}^k(x_0)$ be the running estimate of the gradient for the first k walks at x_0 (computed as in Sec. 3). Since this estimate is unbiased, and the linear term $\nabla u(x_0) \cdot (x - x_0)$ in the Taylor series for $u(x_0)$ integrates to zero over any ball around x_0 (*i.e.*, $c = 0$), we can replace any existing WoS estimator $\widehat{u}$ with the estimate

$$\frac{1}{N} \sum_{i=1}^{N} \widehat{u}(x_0) - \overline{\nabla u}^{i-1}(x_0) \cdot (x_1^i - x_0), \qquad x_1^i \sim \mathcal{U}(B(x_0)), \tag{16}$$

where x_1^i denotes the first step in the ith walk. (Note also that the estimate $\overline{\nabla u}^{i-1}$ is statistically independent of the ith walk.) In practice even this simple strategy can help reduce variance—see for example Fig. 6.

4.1.2 Control Variate for Gradient. Variance reduction for derivatives is especially important, since differentiation amplifies high frequencies. Our control variate strategy for the gradient is complementary to the one for the solution: let $\overline{u_0}^k(x_0)$ be the running estimate of the solution $u_0(x_0)$ for the first k walks. Then we can replace the gradient estimator for the boundary term (Eqn. 13) with

$$\frac{n}{R} \frac{1}{N} \sum_{i=1}^{N} (\widehat{u}_0(x_0) - \overline{u}_0^{i-1}(x_0)) \nu(x_1^i).$$

Since (by symmetry) the normal ν integrates to zero over the sphere, the expected value of the estimator is unchanged ($c = 0$) but the variance is typically lower since the control variate gets closer and closer to the true value of $u_0(x_0)$. Likewise, if $\overline{f}^k := \frac{1}{k} \sum_{i=1}^{k} f(y_0^i)$ is the running average of source values sampled from the initial ball, then we can subtract this value from $f(y)$ in our estimator for the initial source term in Eqn. 14. This strategy parallels methods used in reinforcement learning [Sutton and Barto 2018, Section 13.4], and is related to techniques used on discrete grids [Newton 1994].

4.1.3 Control Variate for Hessian. This approach extends to higher-order derivatives. For example, in the ith term of our Hessian estimator for the boundary term (Eqn. 15) we can replace $\widehat{u}_0(x_1^i)$ with

$$\widehat{u}_0(x_1^i) - \overline{u}_0^{i-1}(x_0) - \overline{\nabla u_0}^{i-1}(x_0) \cdot (x_1^i - x_0).$$

(Alanko and Avellaneda [2013] discuss a similar approach for grids.) Note that both running sums have already been computed for the source and gradient control variates; these could now be further improved via the Hessian estimate, though we do not do so. Fig. 6 shows the effect on variance, which is about 7x lower than the baseline estimator from Eqn. 15 (see Fig. 15, *right*).

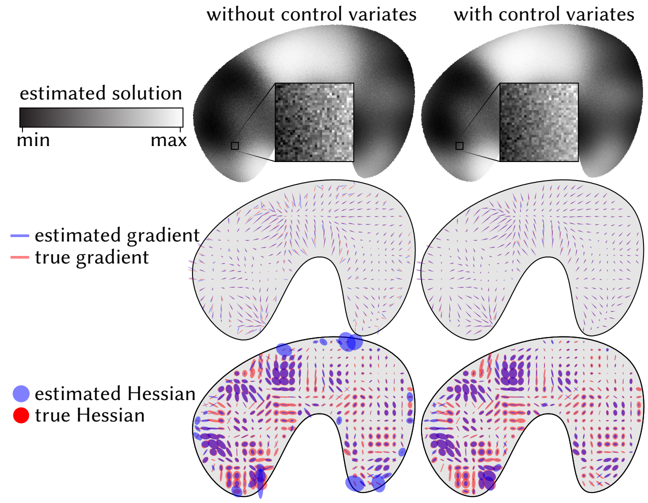


Fig. 6. Control variates provide modest variance reduction for the solution *(top)*, and become more important for higher derivatives of the solution *(middle, bottom)*. Here we plot the Hessian's principal values and axes as ellipses; notice that variance is higher near the boundary, and axes are harder to estimate in regions where the Hessian has small magnitude.

4.2 Importance Sampling

We can also reduce variance via importance sampling (Eqn. 2). Here, sampling the Green's function G or the source term f is analogous to sampling materials or illumination (*resp.*) in Monte Carlo rendering; both strategies can be combined via *multiple importance sampling* [Veach and Guibas 1995b], as in Fig. 7. We also give a strategy for importance sampling the space of walks in the case of nested equations (Sec. 4.3), akin to bidirectional path tracing [Lafortune and Willems 1993; Veach and Guibas 1995a].

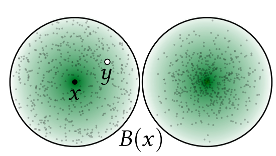


Fig. 8. Uniform *(left)* vs. importance sampling *(right)* the harmonic Green's function.

4.2.1 Green's Functions. Recall that the integral formula for the Poisson equation (Eqn. 7) involves a term $\int_{B(x)} f(y)G(x,y)\,dy$, where G is the harmonic Green's function on $B(x)$. One way to importance sample this term is to simply draw y_k from the distribution $p_G := G / \int_{B(x)} G(x,y)\,dy$. Such samples can be generated via any standard technique (*e.g.*, rejection sampling); in 3D we use *Ulrich's polar method* [Devroye 1986, Section 9.4]. The same strategy can be applied to any PDE with a known Green's function—App. B gives sampling densities for all PDEs appearing in this paper. In principle, one could also extend this strategy to PDEs where the Green's function can only be tabulated numerically, akin to importance sampling for measured BRDFs [Lawrence et al. 2004].

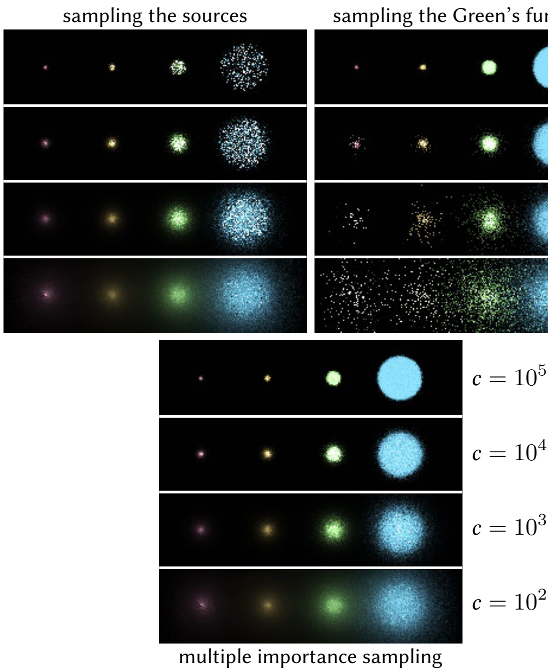


Fig. 7. The source term f and Green's function G play the same role as lights and materials in Monte Carlo rendering (*resp.*); variance can hence be reduced via familiar importance sampling strategies—here we use multiple importance sampling to robustly sample screened Poisson equations for varying coefficients c, with area sources of varying size.

4.2.2 Source Terms. We can also sample the source term f. An important case is a *point source* $f_z := c\delta_z$, where $c \in \mathbb{R}$ is a constant, and δ_z is the Dirac delta centered at a point $z \in \Omega$. Point sources are analogous to point lights in rendering, and appear in applications such as computing shape signatures [Sun et al. 2009] or distances [Crane et al. 2017]. If there is a single source δ_z inside the current ball $B(x)$, we can use an importance density $p = \delta_z$, yielding an importance sampled estimator

$$\frac{f(z)G(x,z)}{p(z)} = \frac{c\delta_z G(x,z)}{\delta_z} = cG(x,z),$$

i.e., just use a single sample at $y = z$. Unlike rendering point lights, there is no visibility term since $B(x)$ is contained entirely in Ω; hence, z can always be reached via random walks from x.

Similarly, consider a *curve source* $f_\gamma := c\delta_\gamma$, where δ_γ is the (1-Hausdorff) measure of a curve $\gamma \subset \Omega$, and c is a function along γ. When γ intersects $B(x)$, we can sample points $y_1, \ldots, y_M$ uniformly from $\gamma_B := \gamma \cap B(x)$ and use the estimator

$$|\gamma_B| \frac{1}{M} \sum_{j=1}^{M} c(y_j) G(x, y_j).$$

Alternatively, one can just uniformly sample the whole curve γ, and drop the contribution of points outside $B(x)$. This strategy is easily generalized to any m-dimensional subset; Fig. 9 shows an example.

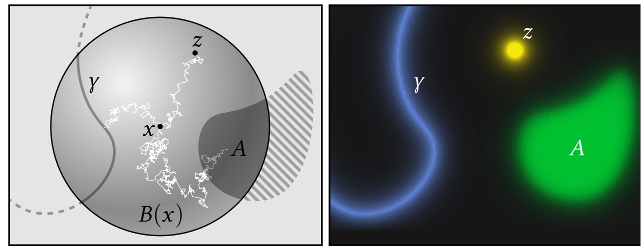


Fig. 9. Source terms can be importance sampled by randomly picking points on the source, restricted to the current ball in the walk. Without importance sampling, only the area source f_A would appear on the right; the point source f_z and curve source f_γ would never get sampled.

4.3 Nested Equations

Our naïve estimator for the biharmonic equation (Sec. 2.5) takes $O(SN^2)$ steps, where N is the number of walks used for the Poisson/Laplace estimators, and each walk takes $O(S)$ steps. We can reduce the cost to $O(SN)$ by introducing a *tree walking* strategy, which increases efficiency by re-using partial walks—in the spirit of bidirectional path tracing (BDPT). Unlike BDPT, we do not generate walks starting at the boundary, but instead connect source samples to the path used for the boundary estimate. More precisely, at each step x_i of the "outer" walk we still use a single point y_i to sample the source term (*à la* Sec. 4.2). But rather than using the basic WoS estimator for $v(y_i)$ (Eqn. 5), we use the estimate $\widehat{v}(y_i) = h(\overline{x}_k)$, where x_k is the final point in the walk used to estimate $u(x_0)$. In other words, we connect a walk of length one (from y_i to x_i) to the longer walk from x_i to x_k, then grab the boundary value (Fig. 10). Though the walks are now more strongly correlated, they are also significantly cheaper to compute—in practice, we get reduced variance for equal compute time (Fig. 11). The same kind of thinking can in principle be applied to other PDEs; it may also be interesting to consider connecting walks of different lengths (as in BDPT).

5 SOLVER FRAMEWORK

The utility of WoS for geometry processing can be greatly enhanced via intelligent treatment of boundary representations (Sec. 5.1) and visualization (Sec. 5.2). Just as open source software like PBRT [Pharr et al. 2016] and `Mitsuba` [Jakob 2010] has played an important role

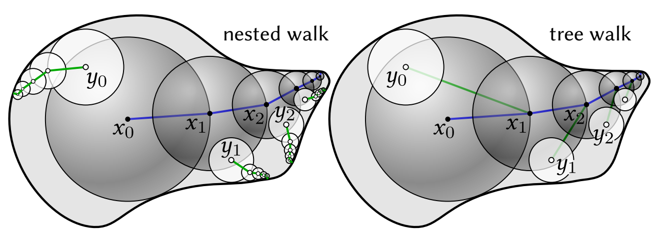


Fig. 10. *Left:* naïvely nesting a Poisson and Laplace estimator to solve a biharmonic equation yields a large number of walks. *Right:* by re-using the boundary path (blue) to compute source contributions (green), we obtain lower variance for the same total compute time.

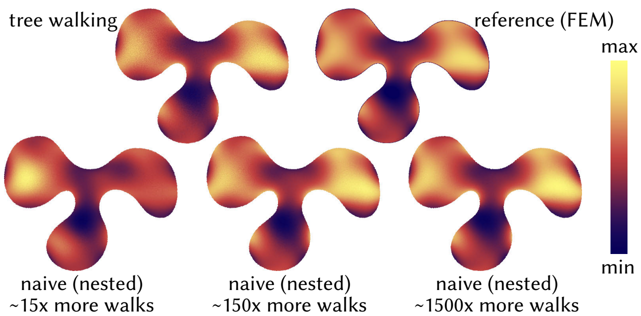


Fig. 11. Here we compare our integration strategy for biharmonic equations to a naïve nested strategy with the same number of outer walks, and 1, 10, and 100 inner walks; for this example each outer walk takes about 15 steps. The tree walking result is slightly noisier, but many times more efficient.

in rendering research, we have built these features into an open-source PDE solver called `zombie` (in homage to "random walkers"). The design of `zombie` is based on `PBRT`, but objects from rendering have been replaced with appropriate analogues from PDE theory. All code was written from scratch in C++. We currently use only basic CPU thread-level parallelism; significant performance gains can likely be made via vectorization and GPU acceleration.

5.1 Boundary Representations

To find an empty ball $B(x)$ around a given point $x \in \Omega$, we need only determine the distance R to the closest point $\overline{x} \in \partial\Omega$, or a conservative underestimate of this distance. As detailed here, such distances are easily computed for a wide variety of shapes; multiple shapes can be combined by taking the minimum over all per-shape distances, or more generally, by applying Boolean operations (Sec. 5.1.4).

Closest Points. Closest point queries can be accelerated via a spatial hierarchy [Pharr et al. 2016, Chapter 4]. Relative to ray tracing [Parker et al. 2010; Wald et al. 2014], there has been very little work on high-performance closest point queries [Shellshear and Ytterlid 2014; Ytterlid and Shellshear 2015]. We currently just use a basic axis-aligned non-vectorized BVH, though recent GPUs provide opportunities for massive acceleration [Wald et al. 2019].

5.1.1 Polygon Soup. Real-world geometry is often given as a list of polygons without explicit connectivity, and which satisfies no special conditions (manifold, orientable, *etc.*). We can solve PDEs directly on such "polygon soup" by simply taking the closest point among all polygons. Rather than attempt to fix cracks, holes, and self-intersections [Shen et al. 2005; Attene et al. 2013], we simply solve an exterior problem *à la* Sec. 2.6 (Fig. 2, *right*). Note that in contrast to generalized winding numbers [Jacobson et al. 2013], the input need not meet any special conditions on orientation.

5.1.2 Parametric Curves and Surfaces. In 2D we use the method of Chen et al. [2007] to compute closest points on cubic Bézier curves. Such a representation is attractive for 2D illustration tools (such as *Illustrator* or *Inkscape*), since it avoids mesh generation and quantization error (consider Fig. 17). Closest points can also be computed directly for NURBS and subdivision surfaces [Dyllong and Luther 2000; Ullrich et al. 2007], but are not yet implemented in our framework.

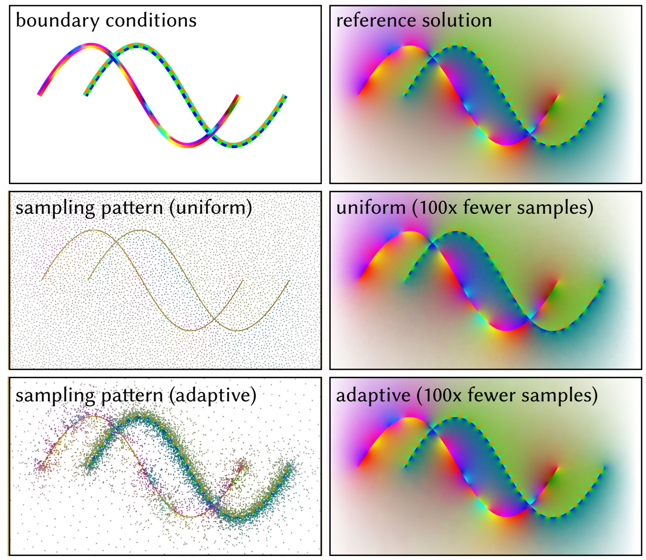


Fig. 12. Just as mesh-based methods interpolate solution values at a few sparse points, we can rapidly visualize solutions to PDEs via scattered data interpolation. Here we solve a Laplace problem using either uniform or adaptive sampling and simple MLS interpolation to reduce the sample cost by 100x. Notice that adaptive sampling better resolves the high-frequency boundary conditions.

5.1.3 Implicit Surfaces. Many shapes are concisely described by the zero level set of a function $\phi : \mathbb{R}^n \to \mathbb{R}$. When ϕ is a signed distance function, the size of the largest ball around x is simply $|\phi(x)|$. More generally, conservative estimates of the distance to an implicit surface can be obtained by bounding the gradient $|\nabla\phi|$, which in turn gives a Lipschitz constant for ϕ [Hart 1996, Theorem 1]. A rich variety of shapes can be described this way [Hart 1996, Table 1]; Fig. 2, *left* shows a smooth blend between two tori.

5.1.4 Booleans. Boolean operations can be used to concisely encode complex models, *à la constructive solid geometry (CSG)* [Requicha and Voelcker 1977]. Tremendous effort has been put into developing robust *mesh booleans* [Bernstein and Fussell 2009; Pavić et al. 2010; Zhou et al. 2016], but they generally remain expensive and error prone, cannot be mixed with other geometric representations, and still require meshing of the domain interior. In contrast, ray tracing can evaluate booleans via simple arithmetic on intersection distances [Roth 1982]. We can likewise combine closest point distances to solve PDEs directly on boolean arrangements (Fig. 2, *center*). Operations on distances needed for both hard booleans as well as soft "blends" are detailed in Hart [1996, Table 1].

5.2 Visualization

A unique feature of Monte Carlo it is that it can evaluate the solution at arbitrary points without performing a global solve. We explore how this feature can improve fidelity and reduce end-to-end cost.

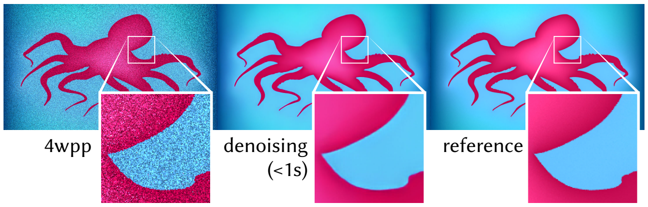


Fig. 13. Even for a small number of walks per pixel *(left)*, techniques for denoising renders *(center)* closely match the reference solution *(right)*, further increasing efficiency—especially for PDEs with smooth solutions.

5.2.1 Interpolation. Elliptic PDEs have highly regular solutions away from the source and boundary. Conventional methods exploit this behavior by interpolating over a mesh; we can likewise interpolate over scattered samples to dramatically reduce cost. For instance, in Fig. 12, *(middle)* we use simple Poisson disk sampling [Bridson 2007] and *moving least squares (MLS)* interpolation [Nealen 2004]. To avoid "bleeding" artifacts we shoot rays to exclude neighbors not visible from sample points. There are plenty of opportunities for improvement and acceleration (*e.g.*, via fast multipole methods [Sun et al. 2014]); note that *all* interpolation schemes introduce bias (including standard nodal interpolation in FEM).

5.2.2 Adaptive Sampling. We can also use adaptive sampling to concentrate effort on interesting regions. For instance, Fig. 12 *(bottom)* applies a simple scheme in the spirit of *irradiance gradients* [Ward and Heckbert 1992]: we first estimate the solution and gradient at a set of seed points. Then, for any candidate sample x, we evaluate a 1st-order Taylor approximation $\widehat{u}(y_i) + \langle \widehat{\nabla u}(y_i), x - y_i \rangle$ for the k nearest neighbors $y_1, \ldots, y_k$ of x. If the sample variance divided by the mean is above a threshold $\sigma > 0$, we estimate the value and gradient at x and add it to the set (we use $\sigma = 0.01$). More sophisticated strategies from rendering provide ample inspiration for future work [Zwicker et al. 2015].

5.2.3 Progressive Rendering. As in rendering, we can progressively increase either the resolution, or the number of walks per sample, to generate a fast "preview" that can be subsequently refined. This approach provides a fast iteration cycle (*e.g.*, interactively adjusting geometry or boundary conditions), and is especially valuable when working with massive models (Fig. 1, *right*).

5.2.4 Denoising. Techniques for denoising rendered images [Rousselle et al. 2013; Kalantari et al. 2015] appear to translate well to the PDE setting. For instance, in Fig. 13 we apply Intel's deep learning-based *Open Image Denoise* algorithm [Intel 2019], using the boundary value at the closest point in place of the albedo map. (Note: no other results in this paper use denoising.) Training such a network on PDE data rather than rendered images may further improve performance.

5.2.5 Region of Interest. Cost can be reduced dramatically by focusing on a small region of interest. For instance, in Fig. 18 we restrict a 3D solution to a 2D slice, reducing cost from $O(n^3)$ to $O(n^2)$ samples. Zooming into a small region of Fig. 16 likewise resolves additional detail without having to refine the entire domain. A more radical cost reduction might be achieved for algorithms that need solution values only at isolated points (*e.g.*, Sun et al. [2009]).

5.2.6 Vector Field Visualization. A vector-valued solution X can be visualized using very sparse sampling. For instance, to draw a standard quiver plot, we need only compute one solution value per arrow—rather than solving the PDE over the entire domain (see Figs. 6 and 19). We can also draw streamlines γ, obtained by numerically integrating the ordinary differential equation (ODE) $\frac{d}{dt}\gamma(t) = X(\gamma(t))$ starting at some collection of seed points. Once again, the value of X need only be estimated at the sparse sample points used by the ODE integrator—examples computed via Huen's method are shown in Figs. 19, 20 and 21.

6 EVALUATION AND GEOMETRIC ALGORITHMS

Here we use several synthetic tests as well as a few fundamental geometric algorithms to evaluate the effectiveness of our Monte Carlo solver for problems in digital geometry processing. Application examples in Sections 6.4–6.7 are meant only to highlight the potential of Monte Carlo methods in geometry processing; further exploration and evaluation are left for future work.

6.1 Stopping Tolerance

The only parameter in the walk on spheres algorithm is the stopping tolerance ε (Sec. 2.2.1). Early stopping extends boundary conditions into an ε-neighborhood $\partial\Omega_\varepsilon$, and the solution in turn exhibits a small bias toward these boundary values. Dirichlet boundary conditions are hence still enforced with the given value, though the location of enforcement may be off by a tiny distance ε. Though in principle bias can be completely eliminated via a *Green's function first passage* approach [Given et al. 1997; Mascagni and Hwang 2003], a pragmatic solution is to simply use a small value of ε.

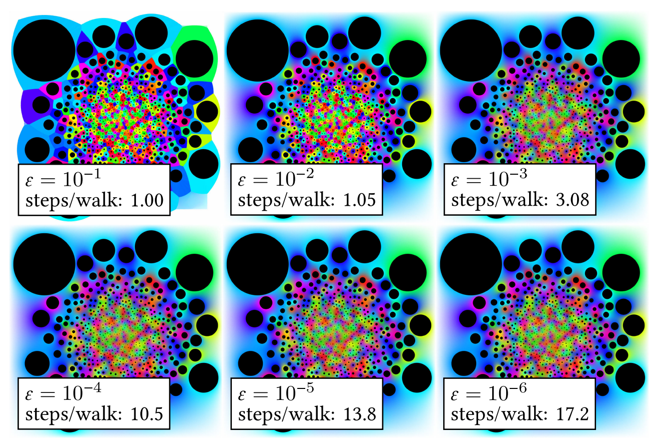


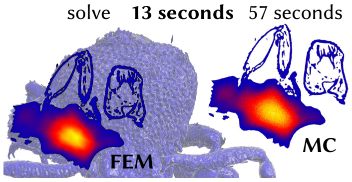
Fig. 14. Tiny features are preserved for any value of ε. For very large values of ε, the walk on spheres algorithm jumps to the closest point, producing a Voronoi-like solution *(top left)*. Decreasing ε quickly eliminates any bias. Since small ε values do not significantly increase the average number of steps per walk, it is generally unnecessary to hand-tune this parameter. (Note that if the box above represents a domain 1 meter in width, then 10^{-4} is about the width of a human hair; small bacteria are on the order of 10^{-6}.)

In practice, an accurate solution is obtained for fairly large ε values—even in the presence of tiny features (Fig. 14). Importantly, arbitrarily small geometric features will *always* appear in the solution, since there will always be evaluation points x in their Voronoi region, independent of ε. Moreover, such features will still have a global effect on the solution, since a random walker has a nonzero probability of reaching any boundary component of finite size. The only potential problem is if spacing *between* features is smaller than ε, which is easily avoided by (universally) using a small ε. Note that shrinking ε by a few orders of magnitude does not significantly increase cost due to the exponential shrinking of balls [Binder and Braverman 2012]. In contrast, mesh-based solvers can eliminate small features entirely (Fig. 27), and their time/memory cost often blows up dramatically with smaller tolerances ε (Sec. 6.4).

6.2 Comparison to Finite Elements

Making an apples-to-apples comparison to FEM is difficult due to both (i) the very different capabilities of the two classes of methods (*i.e.*, different possible inputs and outputs) and (ii) the fact that FEM solver code is very mature, whereas our implementation is an unoptimized research prototype. Comparison with high performance ray Monte Carlo rendering code suggests significant room for improvement. For instance, Intel's *Embree* [Wald et al. 2014] performs about 20–200x more ray queries per second than our code performs closest point queries—even though these two queries have extremely similar structure. Our current implementation is hence much slower than FEM for basic problems (*e.g.*, computing all nodal values on a coarse tet mesh), though when factors such as meshing are taken into account, it is already quite competitive in terms of end-to-end cost (see for instance Fig. 1, and Figs. 18 and 27).

method	linear FEM	Monte Carlo
#triangles	2M	**10M**
#samples	**47k nodes**	23k pixels
precompute	14 hours	**0.4 seconds**
solve	**13 seconds**	57 seconds



To get a rough sense of the tradeoffs, we solved a Poisson equation on the model in Fig. 1 using WoS and piecewise linear FEM (see inset). To use FEM we had to first generate a tetrahedral mesh; the only suitable method (due to many holes and self-intersections in the input) was *FastTetWild*, which takes a geometric tolerance as input. Though the input mesh had 10M triangles, the smallest tolerance our machine could handle (5×10^{-4}) simplified the input to 2M triangles after 14 hours, and hence (as in Fig. 27) destroyed biologically relevant features. The resulting tet mesh had 500k vertices, but only 47k of these were interior nodes—hence, the spatial resolution of the FEM solution was quite poor, even though the tet mesh was rather large. In contrast, our Monte Carlo solver needed only 0.4 seconds for precomputation (to build a BVH), and exactly preserved the input boundary mesh by design. We estimated the solution at a similar number of pixels (23k), providing a solution with superior spatial resolution (and a bit of noise), albeit only on a slice rather than the whole volume. Overall, although the FEM "solve" step is hard to beat in terms of raw speed per sample, the end-to-end quality and performance characteristics are quite different.

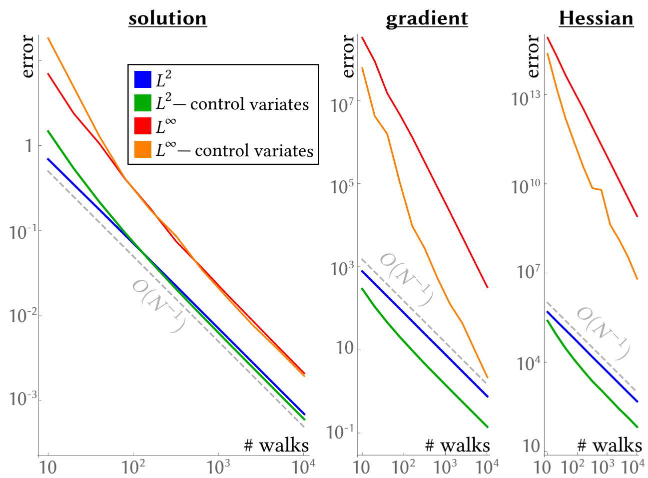


Fig. 15. Since our estimators are unbiased, they exhibit the expected $1/N$ Monte Carlo convergence rate with respect to the number of walks N. Here we plot convergence with respect to the number of walks for the Poisson equation depicted in Fig. 6, showing both the average (L^2) and worst (L^∞) error over about 400 regularly-spaced evaluation points. As walks increase, control variates provide a roughly 5x and 7x reduction in average error for gradients and Hessians, *resp.*

6.3 Convergence

Convergence to the true (smooth) solution is guaranteed simply by the fact that we use unbiased Monte Carlo estimators. More precisely, an N-sample Monte Carlo estimator F_N as described in Eqn. 1 is *unbiased* if the expected value $E(F_N)$ is equal to the value of the original integral (even for $N = 1$). In this case it is also *consistent*, *i.e.*, as $N \to \infty$, the error $F_N - I$ goes to zero with probability one [Veach 1997, Section 2.4.3]. In particular, the variance will decrease at a rate $1/N$, independent of dimension. Fig. 15 verifies that our estimators exhibit the expected convergence behavior, and generally converge faster when adopting the control variate strategy outlined in Sec. 4.1. Note that interpolation (*à la* Sec. 5.2.1) will introduce bias, though consistency can still be provided by schemes like MLS under reasonable conditions on sample distribution [Mirzaei 2015].

6.4 Diffusion Curves and Surfaces

To examine the ability of our solver to handle complex boundaries and boundary conditions, we implemented *diffusion curves* (Fig. 16) and *diffusion surfaces* (Fig. 18), which encode a resolution-independent 2D image or volumetric texture as the solution to a Poisson equation with two-sided boundary conditions. A nice feature of Monte Carlo is that we can directly compute the solution on a zoom-in or cross section by just evaluating points of interest, rather than precomputing a global solution (as in Orzan et al. [2008, Section 3.2.4]). Moreover, the Monte Carlo approach completely decouples signal frequency from geometric resolution—for instance, Fig. 12 shows highly oscillatory boundary conditions on just two

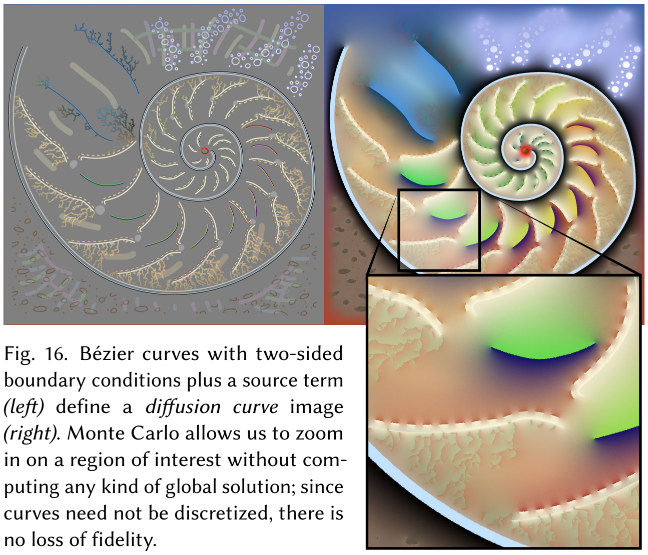


Fig. 16. Bézier curves with two-sided boundary conditions plus a source term *(left)* define a *diffusion curve* image *(right)*. Monte Carlo allows us to zoom in on a region of interest without computing any kind of global solution; since curves need not be discretized, there is no loss of fidelity.

cubic Bézier segments. In contrast, traditional methods must explicitly refine a background grid or mesh to even *encode* such boundary conditions (much less compute a solution).

To further examine tradeoffs with the finite element approach, we generated triangular and tetrahedral meshes (*resp.*) for these two domains using *TriWild* [Hu et al. 2019a] and *FastTetWild* [Hu et al. 2019b], which are state-of-the-art robust meshing algorithms. In addition to needing significant time and memory *just for mesh generation* (*i.e.*, without even solving the PDE), the default tolerances in these methods were not sufficient to capture fine-scale details. Tuning such tolerances is expensive since a mesh must be re-generated each time, and is hard to do in an automatic fashion. Moreover, tolerances that are *too* tight can lead to an explosion in cost—for instance, even just lowering *FastTetWild*'s default tolerance of 10^{-3} to 10^{-4} used up 22GB of memory and failed to generate a mesh after 14 hours of wall clock time.

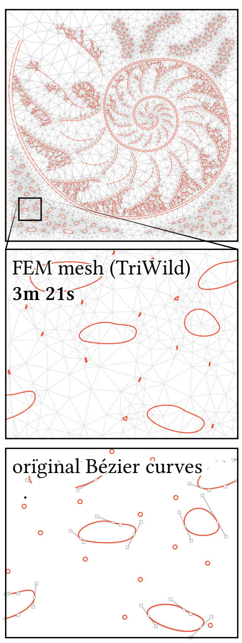


Fig. 17. Even robust meshing algorithms can distort small geometric details.

In contrast, the tolerance ε in the Monte Carlo approach (Sec. 2.2.1) has very little effect on cost and accuracy: for very loose tolerances small features can get rounded out, but cannot disappear (Fig. 14). Even tiny tolerances add only a few steps to each walk, due to the $\log(1/\varepsilon)$ behavior of the walk on spheres algorithm (Fig. 14), *i.e.*, computation cost does not blow up dramatically.

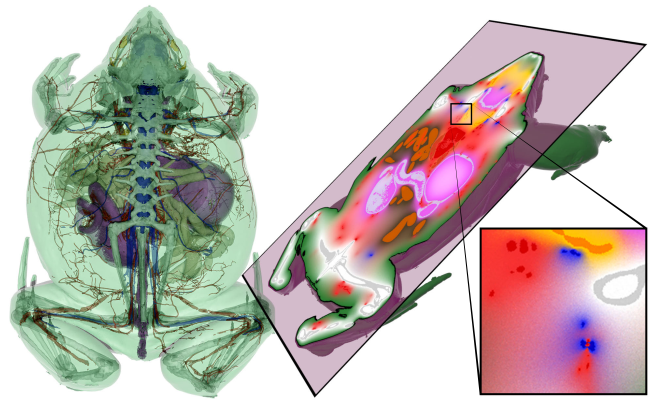


Fig. 18. *Left:* a 3.3M triangle boundary mesh with fine features, from a CT scan of *hemisus guineensis* (courtesy Blackburn Lab). To visualize the solution to a volumetric PDE, we can significantly reduce cost by computing only a 2D slice of the full 3D solution *(right)*. *Inset:* fine features are perfectly preserved since we work with the exact input geometry.

6.5 Symmetric Direction Fields

We also explored the use of our solver in problems with vector-valued solutions. Rotationally symmetric direction fields such as *line fields* and *cross fields* play a key role in algorithms ranging from quadrilateral meshing [Bommes et al. 2013] to texture synthesis [Knöppel et al. 2015]. We implemented a cross field generation scheme that adopts ideas from Knöppel et al. [2013]. Consider a domain $\Omega \in \mathbb{R}^n$, and let $\nu : \partial\Omega \to \mathbb{C}$ denote the outward normals along the boundary, encoded as unit complex numbers. The function $\widetilde{\nu} := \nu^4$ represents a cross field adapted to the boundary: if two normal vectors ν_1, ν_2 differ only by quarter rotations, then $\widetilde{\nu}_1 = \widetilde{\nu}_2$. To get the smoothest cross field compatible with this boundary cross field, we solve the Laplace equation $\Delta\psi = 0$ subject to Dirichlet boundary conditions $\psi|_{\partial\Omega} = \nu^4$ for a function $\psi : \Omega \to \mathbb{C}$. At any given point x, the fourth roots of the normalized solution value $\varphi(x) := \psi(x)/|\psi(x)|$ then give the four directions of the cross. The only change to our basic Laplace estimator (Sec. 2.2) is that we accumulate complex values rather than real ones.

To visualize the solution, we need only solve for ψ at the (sparse) set of points where we wish to draw crosses. Following Viertel and Osting [2017], we can also draw integral curves connecting field singularities—we implemented a basic version of this scheme. To locate singularities, we sample a collection of random seed points x_0 and perform gradient descent on the field magnitude $|\nabla\psi|^2$, *i.e.*, we numerically integrate the ODE

$$\tfrac{d}{dt}x = -2\langle\psi, \nabla\psi\rangle,$$

where $\nabla\psi$ is evaluated via Eqn. 12. The index $n \in \mathbb{Z}$ of the final (singular) point x^* can be determined by picking any point x near x^* and evaluating the expression

$$n = -\imath\frac{|x - x^*|}{\psi(x)} D_u\psi(x),$$

where D_u denotes the derivative along the direction $u := \imath(x - x_0)/|x - x_0|$. This value will approach a real integer as $x \to x^*$.

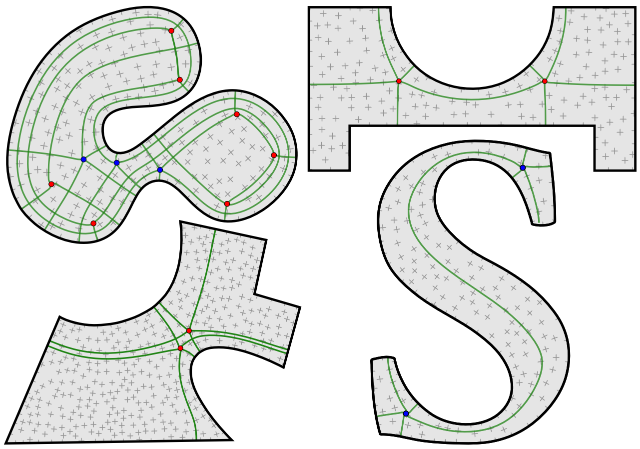


Fig. 19. Cross fields and corresponding quad layouts obtained by connecting singular points; no meshes were used at any stage.

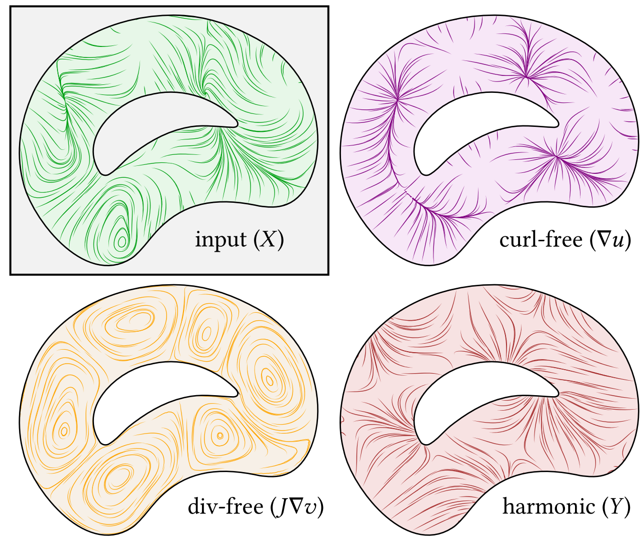


Fig. 20. With the Monte Carlo approach, one does not even have to discretize input data. Here we decompose an input vector field X given by a closed-form, analytic expression on a domain with a Bézier boundary curve (top left) into its curl-free, divergence-free, and harmonic components.

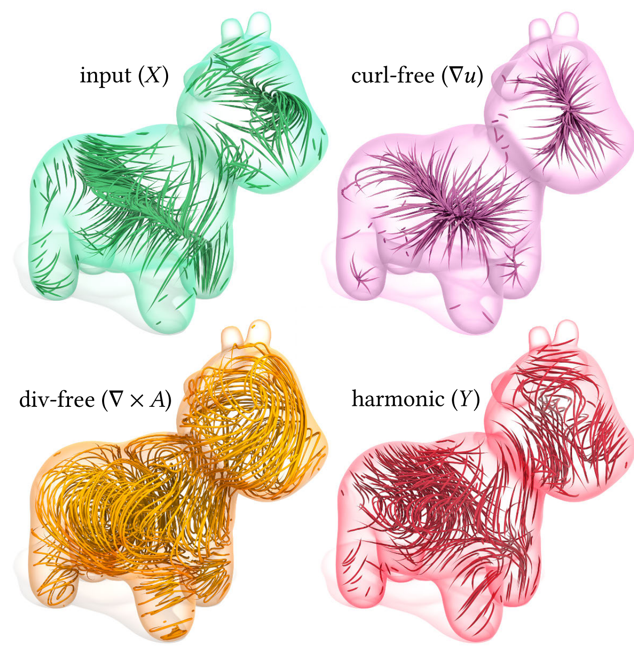


Fig. 21. Helmholtz decomposition on a 3D domain. The Monte Carlo approach entirely avoids meshing the domain, setting up matrices, and solving linear systems.

Integral curves are then traced from the singularities as described in Sec. 5.2.6, except that at each step of integration we follow the cross direction best aligned with the curve tangent. Several examples are shown in Fig. 19. Note that unlike mesh-based methods, we never have to worry about spurious singularities arising from principal matchings (as discussed by Vaxman et al. [2016, Section 6.3]). Moreover, since we are ultimately just taking an average of boundary values with respect to the harmonic measure, this scheme opens the door to more nonlinear 3D field generation problems—we leave such extensions to future work.

6.6 Helmholtz Decomposition

A powerful tool in flow processing and visualization is the *Helmholtz decomposition*, which expresses a given vector field X as the sum of a curl-free, divergence-free, and harmonic part. In computer graphics, such decompositions have been considered on meshes [Tong et al. 2003; Zhao et al. 2019] and point clouds [Ribeiro et al. 2016], both of which entail discretizing space and solving large linear systems. On a domain $\Omega \subset \mathbb{R}^3$, one possible decomposition is

$$X = \nabla u + \nabla \times A + Y, \tag{17}$$

where u is the solution to the scalar Poisson equation $\Delta u = \nabla \cdot X$, A is the solution to the vector Poisson equation $\Delta A = \nabla \times X$ (which is just three componentwise scalar equations), and Y is the remaining part. Setting u and A to zero along $\partial\Omega$ yields standard *normal-parallel* boundary conditions [Bhatia et al. 2014, Section 4.2], which ensure the decomposition is unique [Bhatia et al. 2013]. In 2D we get an analogous decomposition by replacing A with a scalar potential $a : \Omega \to \mathbb{R}$, and replacing $\nabla \times A$ with $J\nabla a$, *i.e.*, a 90-degree rotation of the gradient of a.

To compute this decomposition via Monte Carlo, we compute the necessary derivatives (Sec. 3) of the solution to the two Poisson equations (Sec. 2.3). Basic 2D and 3D examples are shown in Fig. 20 and Fig. 21. As in Sec. 6.5 we also trace streamlines, again by applying a standard ODE integrator to the derivative estimates.

6.7 Shape Deformation

We implemented a simple 2D shape deformation tool that works directly with Bézier curves, rather than building a cage or mesh (Fig. 22). To do so, we solve a biharmonic equation for a coordinate

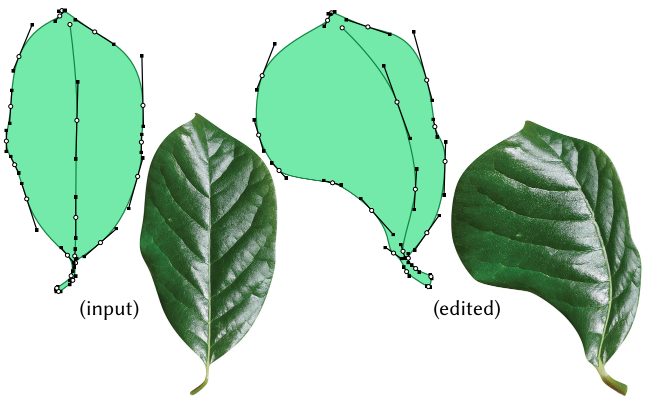
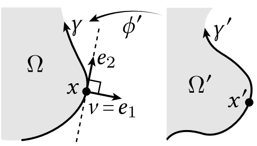


Fig. 22. We can perform shape deformation by directly editing Bézier curves, including open curves on the shape interior.

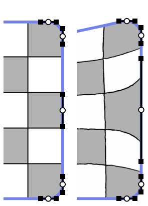


mapping $\phi' : \Omega' \to \Omega$, where Ω and Ω' are the source and target domains, *resp.* We can then use ϕ' to lookup into a texture. The 0th-order boundary conditions at any point $x' \in \partial\Omega'$ are just the corresponding coordinate values $\phi(x)$ on the input domain (which we determine via the Bézier parameterization). Second-order boundary conditions provide "as linear as possible" behavior near the boundary, in the spirit of Stein et al. [2018]. More precisely, we let $e_1 := \nu'$ be the unit normal to $\partial\Omega'$, and enforce the condition $\partial^2\phi'/\partial e_1^2 = 0$, *i.e.*, no second derivative in the normal direction. To do so, we note that $\Delta\phi' = \partial^2\phi'/\partial e_1^2 + \partial^2\phi'/\partial e_2^2$, where e_2 is orthogonal to e_1. Hence, we can just use boundary conditions $\Delta\phi' = \partial^2\phi/\partial e_2^2$ in Eqn. 10 to get a vanishing second normal derivative (since $\phi = \phi'$ along the boundary). Importantly, the second derivative along e_2 is *not* the same as the second derivative along the boundary curve γ, since a curve agrees with its tangent line only up to first order. Using the chain rule one can in fact show that

$$\frac{\partial^2\phi}{\partial e_2^2} = \frac{d^2}{ds^2}(\phi\circ\gamma) + \kappa\nu,$$

where s is the arc-length parameter for γ, κ is its curvature, and ν is its unit normal. Since a Bézier curve $\gamma(t)$ is not typically arc-length parameterized, we use the formula

$$\frac{d^2}{ds^2}(\phi\circ\gamma) = \frac{d^2u/dt^2}{|d\gamma/dt|^2} - \frac{du/dt}{|d\gamma/dt|^4}\left\langle d^2\gamma/dt^2, d\gamma/dt\right\rangle.$$



Derivatives with respect to t (as well as the usual formula for κ) are then easily expressed via the usual Bernstein basis. Relative to triangle meshes, this higher-order representation makes it easy to directly control things like the tangential "stretching" of the boundary (see inset). On the other hand, computing each deformation directly via WoS is dramatically slower than with state-of-the-art FEM techniques [Jacobson et al. 2011]; like these methods, it would be wise to consider how a basis of deformations can be precomputed and re-used. Finally, it is interesting to consider analogous strategies for volumetric deformations, via subdivision or NURBS surfaces (Sec. 5.1.2).

7 RELATED WORK

We here contrast the Monte Carlo approach with more conventional solvers for linear elliptic PDEs (Laplace, Poisson, *etc.*). The literature on solving these problems is vast; here we consider aspects specifically relevant to digital geometry processing.

7.1 Discretization Error

The basic premise of conventional solvers is to reduce a PDE to a finite dimensional system of equations. Constructing this system necessitates sampling or tessellating the domain or its boundary, making discretization error inevitable. In contrast, WoS introduces no spatial discretization—the only opportunity for error is in the stopping tolerance ε, which has virtually no effect on geometric quality (Sec. 6.1). Likewise, conventional methods must refine the mesh to capture detailed boundary conditions (Fig. 24), whereas WoS decouples the resolution of boundary data from the geometry. An analogy from rendering is the use of high-resolution image-based lighting [Debevec 2002], which greatly enhances the richness and flexibility of illumination relative to finite element radiosity [Gershbein et al. 1994].

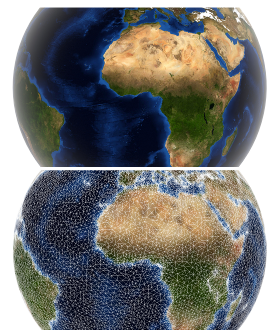


Fig. 24. Even on simple domains, conventional methods must refine the mesh to capture detailed boundary data.

7.2 Approximation Error

Conventional methods must also approximate differential operators, using either local Taylor approximation (as in finite difference methods), or by restricting solutions to a finite-dimensional function space (as in FEM). Proving consistency with smooth solutions is challenging even for fairly common equations [Dziuk 1988; Stein et al. 2019]. Schemes that are consistent in a weak sense can still give unreliable *pointwise* derivatives (Fig. 26), which are needed for tasks like curvature evaluation [Meyer et al. 2003]. Using higher-order elements can significantly inflate degree—on triangle meshes, for instance, 5th-order polynomials are needed to get even C^1 continuity [Papanicolopulos and Zervos 2013], and standard subdivision

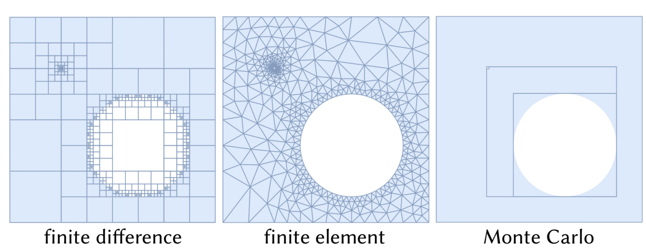


Fig. 23. Grid- and mesh-based algorithms *(left, center)* consume significant memory since they explicitly discretize even simple geometry, and must use gradual *grading* to maintain good element quality. Monte Carlo methods *(right)* concisely represent smooth geometry and small features via a bounding volume hierarchy (BVH), providing excellent memory scaling.

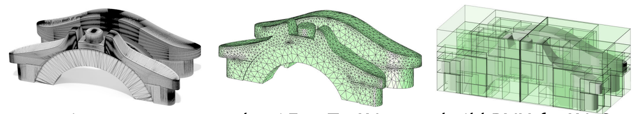


Fig. 25. Finite element methods exhibit unpredictable performance, since models with simple geometry but poor element quality *(left)* can confound even robust meshing algorithms *(center, from Hu et al. [2019b])*. The Monte Carlo approach simply needs to build a standard bounding volume hierarchy *(right)*, dramatically reducing the end-to-end cost of solving a PDE.

bases are not C^2 at irregular vertices [Peters and Reif 2008, Section 1.8]. Moreover, the convergence of FEM can slow down in the presence of features like *reentrant corners* [Givoli et al. 1992, Equation 2], which are common in real-world geometry. Since WoS exactly simulates a stochastic process that models the smooth PDE, such issues simply do not arise.

7.3 Boundary Representations

Conventional solvers must convert high-order or implicit boundary representations (*e.g.*, NURBS or CSG models) into explicit meshes that can be used for analysis. Monte Carlo methods enable geometry processing algorithms to be run directly on a much richer class of representations, such as curved patches (Sec. 5.1). It also works with *implicit* descriptions like algebraic surfaces, and (as in rendering) can achieve extreme geometric complexity at low memory cost via techniques like instancing [Snyder and Barr 1987] or procedural modeling [Deussen et al. 1998]. Particularly relevant for geometry processing are *boolean operations*, which are notoriously difficult to compute explicitly [Bernstein and Fussell 2009; Pavić et al. 2010; Zhou et al. 2016], yet with WoS amounts to simple arithmetic on distance values (Sec. 5.1.4).

7.4 Robustness

Meshes encountered in the wild often exhibit abysmal element quality, are nonmanifold or nonorientable, fail to be watertight, or have many self-intersections [Zhou and Jacobson 2016]. Such defects are especially virulent for finite element mesh generation, where even small defects can induce total failure [Hu et al. 2018, Section 2]. One remedy is to perform mesh repair [Attene et al. 2013], though such methods provide few hard guarantees. Recent mesh generation algorithms provide substantial robustness for FEM [Hu et al. 2018, 2019a,b], but models with poor element quality can be prohibitively expensive to mesh (Fig. 25), and efficiency comes at the cost of spatial quantization that can destroy important features (Fig. 27). Closer to our goal are robust PDE solvers that exactly preserve the input, *e.g.*, by degree elevation [Schneider et al. 2019] or domain decomposition [Sellán et al. 2019]—yet such methods must still face the gauntlet of tetrahedral meshing.

A key feature of grid-free Monte Carlo methods is that solution accuracy has nothing to do with the quality of mesh elements (see Fig. 2, *right*), providing rock-solid "black box" solvers that can be used in existing geometry pipelines. The potential for such tools is hinted at by the method of *generalized winding numbers* [Jacobson et al. 2013], which effectively solves a special case of the Laplace equation with two-sided boundary conditions [Barill et al. 2018, Section 2.1], and has led to robust versions of core geometric algorithms [Zhou et al. 2016; Hu et al. 2018, 2019b]. The grid-free Monte Carlo framework presented here broadens this kind of approach.

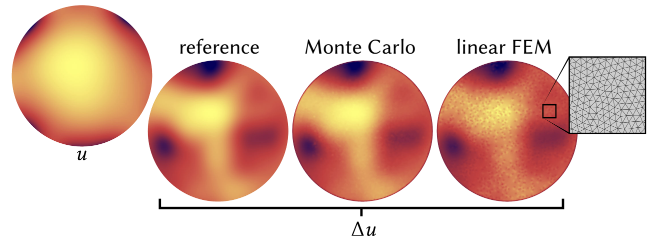


Fig. 26. Even for a smooth solution u on a high-quality (optimal Delaunay) mesh, piecewise linear FEM provides only one weak derivative and cannot be used to reliably evaluate higher-order derivatives like the Laplacian Δu—even if the mesh is refined. Monte Carlo methods provide exact *pointwise* derivatives, in expectation.

7.5 Performance and Scalability

Computationally, the fundamental difference between conventional PDE solvers and Monte Carlo methods is that, in the former, information is shared among nodes (via a linear system), whereas in the latter, values are estimated independently at each point. While information sharing can increase efficiency, it also comes with significant costs—*e.g.*, generating a mesh that connects nodes, or parallel communication overhead.

To compare FEM and Monte Carlo, one must carefully consider the time it takes to obtain a solution of equal resolution and accuracy (Sec. 6.2). Crucially, the basic $O(1/\sqrt{N})$ convergence rate of Monte Carlo provides a measure of error relative to *other methods that estimate integrals* (such as Gauss quadrature)—not those that solve PDEs. Moreover, factors like mesh generation are far more significant in practice than solving the PDE itself. Even state-of-the-art methods such as *FastTetWild* [Hu et al. 2019b] can take many hours or many gigabytes of memory (Figs. 1, 25 and 27). Meshless FEM and boundary element methods (BEM) avoid meshing the interior, but come with their own challenges and tradeoffs (see for instance

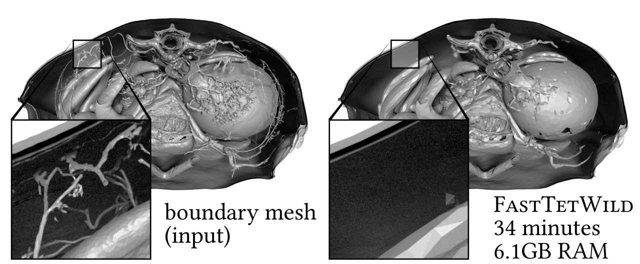


Fig. 27. Robust meshing methods guarantee success at the cost of geometric accuracy—in this case, completely destroying information about the circulatory and central nervous systems from Fig. 18. The Monte Carlo approach exactly preserves these features without hand-tuning of geometric tolerances (nor spending time and memory to precompute a mesh).

[Idelsohn et al. 2003; Costabel 1986]). In particular, BEM struggles to handle source terms, making its use in geometry processing quite limited. Meshfree Galerkin methods [Li and Liu 2007, Chapter 3] replace meshing with a difficult sampling problem where one must adaptively sample fine features while simultaneously maintaining both an upper and lower bound on sample count in each local neighborhood [Li and Liu 2007, 4.1.1]. Without such *admissibility* conditions, meshfree FEM will fail to converge. Moreover, the number of neighbors—and hence system density—must increase as sample spacing decreases [Zhu et al. 2015].

Since WoS computes the solution independently at each point, it avoids mesh generation entirely, and is oblivious to sample distribution. The only precomputation is building a BVH, which has complexity $O(n \log n)$ in the size of the boundary. These savings are compounded for problems with dynamic geometry, where one can just update or re-fit the BVH [Kopta et al. 2012].

Other practical issues have a big impact on performance. For example, Monte Carlo methods are trivial to parallelize, whereas parallel meshing and matrix factorization are notoriously difficult. Monte Carlo can use local evaluation to asymptotically reduce cost (Fig. 18), whereas conventional methods must always perform a global solve—even BEM involves a dense system coupling all boundary degrees of freedom. Conventional adaptive refinement entails remeshing and rebuilding a linear system; with Monte Carlo it is a simple matter of adding more samples (Sec. 5.2.2). Likewise, in real time scenarios where the compute budget is unknown *a priori*, Monte Carlo can just keep sampling, whereas grid-based methods must carefully predict a global resolution ahead of time.

8 LIMITATIONS AND FUTURE WORK

The goal of this paper was to set the groundwork for Monte Carlo Geometry Processing by fleshing out some basic estimators and algorithms; clearly a great deal more could be done. One is to develop good estimators for a larger set of PDEs, which would in turn enable an even larger set of applications. In this paper we restricted our focus to the WoS algorithm because it exhibits no bias or discretization error—though often requires explicit knowledge of certain distributions, *e.g.*, Green's functions. A much broader class of stochastic methods could be used to enrich the set of available PDEs [Higham 2001; Kloeden and Platen 2013]. While these methods sometimes introduce (say) a controlled amount of discretization error, they retain many of the perks of Monte Carlo methods (scalability, geometric flexibility, output sensitivity, *etc.*). In particular, PDEs with spatially inhomogeneous coefficients can be solved in the WoS framework via walks with smaller steps (akin to volumetric pathtracing). Some nonlinear PDEs can likewise be handled by, *e.g.*, simulating *branching diffusion* [Bossy et al. 2015], or by applying *forward-backward stochastic differential equations* [Pardoux and Tang 1999]. We also did not treat a variety of boundary conditions (Neumann, Robin, *etc.*) which often show up in applications—such conditions might be incorporated using either WoS [Maire and Tanré 2013] or other stochastic methods.

On the geometric side, the most glaring omission is that we do not treat surface (*i.e.*, shell) problems—the basic reason is that on a surface with inhomogeneous curvature the statistical assumption of WoS no longer holds, *i.e.*, the exit location of a random walk starting from the center of a ball is not uniformly distributed over the ball's boundary. Here it might be interesting to explore a polyhedral WoS algorithm (since polyhedral surfaces are at least locally Euclidean). It would also be beneficial to explore subdivision and especially NURBS boundary representations, which could help avoid meshing challenges in common engineering/analysis applications.

There are many opportunities for performance improvements—for instance, building a high-performance closest point library (as hinted at in Sec. 6.2), or developing a GPU implementation. Spheres in the WoS algorithm can be replaced by other sets, such as rectangular prisms [Deaconu and Lejay 2006], which can help increase efficiency for, *e.g.*, domains with thin features. Many possible variance reduction strategies were not explored—for instance, importance sampling boundary data, which might be achieved via *drifted random walks* [Øksendal 2003, Section 8.6]. Note also that the Hessian estimator described in Sec. 3.3 can only be used in problems where the gradient of the source function f is known; an estimator for "black box" functions f might be obtained by importance sampling the Hessian of the Green's function. Finally, it seems wise to consider hybrid strategies that combine Monte Carlo estimation with information sharing (see discussion in Sec. 7.5), perhaps by way of multi-level schemes [Heinrich 2001] or temporal difference methods in reinforcement learning [Sutton and Barto 2018, Chapter 6].

In the long run, a likely outcome is that there are problems and algorithms from the FEM setting that do not naturally translate to Monte Carlo methods, and likewise, paths that can easily be taken via Monte Carlo but not via traditional PDE solvers. For instance, we did not even touch on the possibility of solving problems with unknown, random variables (as often arise when working with measured data), which seems quite natural in the Monte Carlo setting. On the whole, we are optimistic that the Monte Carlo approach to PDEs will lead to creative new approaches—and problems—in geometry processing, and look forward to seeing how interactions between geometry, rendering, and stochastic calculus can continue to fortify this effort.

ACKNOWLEDGMENTS

The authors thank Mark Gillespie, Ioannis Gkioulekas, and Rasmus Tamstorf for fruitful conversations, Ruihao Ye for support on closest point queries, and Timothy Sun for some early explorations of these ideas. The hemisus guineensis mesh (Fig. 18) is used courtesy of the Blackburn Lab, under a CreativeCommons BY 4.0 license. This work was supported by a Packard Fellowship, NSF awards 1717320 and 1943123, and gifts from Autodesk, Adobe, Disney, and Facebook. The second author was also supported by NSF award DMS-1439786 and Sloan award G-2019-11406 while in residence at ICERM.

REFERENCES

S. Alanko and M. Avellaneda. 2013. Reducing variance in the numerical solution of BSDEs. *Comptes Rendus Mathematique* 351, 3-4 (2013), 135–138.
J. Arvo. 2001. Stratified sampling of 2-manifolds. *SIGGRAPH Course Notes* 29, 2 (2001).
M. Attene, M. Campen, and L. Kobbelt. 2013. Polygon mesh repairing: An application perspective. *ACM Computing Surveys (CSUR)* 45, 2 (2013), 15.
S. Axler, P. Bourdon, and R. Wade. 2013. *Harmonic function theory*. Vol. 137. Springer Science & Business Media.
G. Barill, N. G Dickson, R. Schmidt, D. Levin, and A. Jacobson. 2018. Fast winding numbers for soups and clouds. *ACM Trans. Graph.* 37, 4 (2018), 1–12.

D. Bell. 2012. *The Malliavin Calculus.* Courier Corporation.
G. Bernstein and D. Fussell. 2009. Fast, exact, linear booleans. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 1269–1278.
H. Bhatia, G. Norgard, V. Pascucci, and P. Bremer. 2013. The Helmholtz-Hodge Decomposition—A Survey. *IEEE Trans. Viz. Comp. Graph.* 19, 08 (2013).
H. Bhatia, V. Pascucci, and P. Bremer. 2014. The Natural Helmholtz-Hodge Decomposition for Open-boundary Flow Analysis. *IEEE Trans. Viz. Comp. Graph.* 20, 11 (2014).
I. Binder and M. Braverman. 2012. The rate of convergence of the Walk on Spheres Algorithm. *Geometric and Functional Analysis* 22, 3 (01 Jun 2012), 558–587. https://doi.org/10.1007/s00039-012-0161-z
D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. 2013. Quad-mesh generation and processing: A survey. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 51–76.
T. Booth. 1981. Exact Monte Carlo solution of elliptic partial differential equations. *J. Comput. Phys.* 39, 2 (1981), 396–404.
M. Bossy, N. Champagnat, H. Leman, S. Maire, L. Violeau, and M. Yvinec. 2015. Monte Carlo Methods for Linear and Non-linear Poisson-Boltzmann Equation. *ESAIM: Proceedings and Surveys* 48 (2015), 420–446.
R. Bridson. 2007. Fast Poisson Disk Sampling in Arbitrary Dimensions. In *SIGGRAPH sketches.* 22.
X. Chen, Y. Zhou, Z. Shu, H. Su, and J. Paul. 2007. Improved Algebraic Algorithm on Point projection for B´ eziercurves. In *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*. IEEE, 158–163.
M. Chuang and M. Kazhdan. 2011. Interactive and anisotropic geometry processing using the screened Poisson equation. *ACM Trans. Graph.* 30, 4 (2011), 57.
M. Costabel. 1986. *Principles of Boundary Element Methods.* Techn. Hochsch., Fachbereich Mathematik.
Y. Le Coz and R. Iverson. 1992. A stochastic algorithm for high speed capacitance extraction in integrated circuits. *Solid-State Electronics* 35, 7 (1992), 1005 – 1012.
K. Crane, C. Weischedel, and M. Wardetzky. 2017. The Heat Method for Distance Computation. *Commun. ACM* 60, 11 (Oct. 2017), 90–99.
M. Deaconu and A. Lejay. 2006. A Random Walk on Rectangles Algorithm. *Methodology and Computing in Applied Probability* 8, 1 (12 May 2006), 135. https://doi.org/10.1007/s11009-006-7292-3
P. Debevec. 2002. Image-based Lighting. *IEEE Computer Graphics and Applications* 22, 2 (2002), 26–34.
J. Delaurentis and L. Romero. 1990. A Monte Carlo method for Poisson's equation. *J. Comput. Phys.* 90, 1 (1990), 123 – 140.
O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. 1998. Realistic Modeling and Rendering of Plant Ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques.* ACM, 275–286.
L. Devroye. 1986. Sample-based non-uniform random variate generation. In *Proceedings of the 18th conference on Winter simulation.* ACM, 260–265.
P. Dutre, P. Bekaert, and K. Bala. 2006. *Advanced global illumination.* AK Peters/CRC Press.
E. Dyllong and W. Luther. 2000. *Distance Calculation Between a Point and a NURBS Surface.* Technical Report. Duisburg University.
G. Dziuk. 1988. Finite elements for the Beltrami operator on arbitrary surfaces. In *Partial differential equations and calculus of variations.* Springer, 142–155.
B. Elepov and G. Mikhailov. 1969. Solution of the Dirichlet Problem for the Equation $\Delta u - cu = -q$ by a Model of "Walks on Spheres". *U. S. S. R. Comput. Math. and Math. Phys.* 9, 3 (1969), 194–204.
I. Georgiev, T. Ize, M. Farnsworth, R. Montoya-Vozmediano, A. King, B. Lommel, A. Jimenez, O. Anson, S. Ogaki, E. Johnston, A. Herubel, D. Russell, F. Servant, and M. Fajardo. 2018. Arnold: A Brute-Force Production Path Tracer. *ACM Trans. Graph.* 37, 3, Article 32 (Aug. 2018), 12 pages.
R. Gershbein, P. Schröder, and P. Hanrahan. 1994. Textures and radiosity: Controlling emission and reflection with texture maps. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques.* ACM, 51–58.
J. Given, J. Hubbard, and J. Douglas. 1997. A first-passage algorithm for the hydrodynamic friction and diffusion-limited reaction rate of macromolecules. *The Journal of chemical physics* 106, 9 (1997), 3761–3771.
D. Givoli, L. Rivkin, and J. Keller. 1992. A finite element method for domains with corners. *International journal for numerical methods in engineering* 35, 6 (1992).
C. Goral, K. Torrance, D. Greenberg, and B. Battaile. 1984. Modeling the interaction of light between diffuse surfaces. In *ACM SIGGRAPH computer graphics*, Vol. 18. ACM.
J. Hart. 1996. Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer* 12, 10 (1996), 527–545.
Stefan Heinrich. 2001. Multilevel Monte Carlo Methods. In *Proceedings of the Third International Conference on Large-Scale Scientific Computing-Revised Papers (LSSC '01)*. Springer-Verlag, Berlin, Heidelberg, 58–67.
D. Higham. 2001. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM review* 43, 3 (2001), 525–546.
Y. Hu, T. Schneider, X. Gao, Q. Zhou, A. Jacobson, D. Zorin, and D. Panozzo. 2019a. TriWild: Robust Triangulation with Curve Constraints. *ACM Trans. Graph.* 38, 4 (2019), 52.
Y. Hu, T. Schneider, B. Wang, D. Zorin, and D. Panozzo. 2019b. Fast Tetrahedral Meshing in the Wild. *arXiv preprint arXiv:1908.03581* (2019).
Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, and D. Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 37, 4 (2018), 60–1.
C. Hwang, J. Given, and M. Mascagni. 2000. On the rapid estimation of permeability for porous media using Brownian motion paths. *Physics of Fluids* 12, 7 (2000), 1699–1709.
C. Hwang and M. Mascagni. 2004. Electrical capacitance of the unit cube. *Journal of applied physics* 95, 7 (2004), 3798–3802.
S. Idelsohn, E. Onate, N. Calvo, and F. Del Pin. 2003. The meshless finite element method. *Internat. J. Numer. Methods Engrg.* 58, 6 (2003), 893–912.
Intel. 2019. Open Image Denoise. https://openimagedenoise.github.io/.
A. Jacobson, I. Baran, J. Popovic, and O. Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4 (2011), 78.
A. Jacobson, L. Kavan, and O. Sorkine-Hornung. 2013. Robust Inside-Outside Segmentation using Generalized Winding Numbers. *ACM Trans. Graph.* 32, 4 (2013).
W. Jakob. 2010. Mitsuba Renderer. http://www.mitsuba-renderer.org.
J. Kajiya. 1986. The Rendering Equation. In *ACM SIGGRAPH computer graphics*, Vol. 20. ACM, 143–150.
S. Kakutani. 1944. Two-dimensional Brownian Motion and Harmonic Functions. *Proceedings of the Imperial Academy* 20, 10 (1944), 706–714.
N. Kalantari, S. Bako, and P. Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *ACM Trans. Graph.* 34, 4 (2015), 122–1.
M. Kazhdan, M. Bolitho, and H. Hoppe. 2006. Poisson Surface Reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Vol. 7.
P. Kloeden and E. Platen. 2013. *Numerical Solution of Stochastic Differential Equations.* Vol. 23. Springer Science & Business Media.
F. Knöppel, K. Crane, U. Pinkall, and P. Schröder. 2013. Globally optimal direction fields. *ACM Trans. Graph.* 32, 4 (2013).
F. Knöppel, K. Crane, U. Pinkall, and P. Schröder. 2015. Stripe Patterns on Surfaces. *ACM Trans. Graph.* 34, 4 (2015).
D. Kopta, T. Ize, J. Spjut, E. Brunvand, A. Davis, and A. Kensler. 2012. Fast, effective BVH updates for animated scenes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.* ACM, 197–204.
N. Koshlyakov, M. Smirnov, and E. Gliner. 1964. Differential Equations of Mathematical Physics. (1964).
E. Lafortune and Y. Willems. 1993. Bi-Directional Path Tracing. *Compugraphics* (December 1993), 145–153.
J. Lawrence, S. Rusinkiewicz, and R. Ramamoorthi. 2004. Efficient BRDF importance sampling using a factored representation. In *ACM Trans. Graph.*, Vol. 23. ACM.
S. Li and W. Liu. 2007. *Meshfree Particle Methods.* Springer Publishing Company, Incorporated.
Y. Lipman, R. Rustamov, and T. Funkhouser. 2010. Biharmonic Distance. *ACM Trans. Graph.* 29, 3 (June 2010).
S. Maire and E. Tanré. 2013. Monte Carlo approximations of the Neumann problem. *Monte Carlo Methods and Applications* 19, 3 (2013), 201–236.
M. Mascagni and C. Hwang. 2003. ϵ-Shell error analysis for "Walk On Spheres" algorithms. *Mathematics and computers in simulation* 63, 2 (2003), 93–104.
M. Mascagni and N. Simonov. 2004. Monte Carlo Methods for Calculating some Physical Properties of Large Molecules. *SIAM journal on scientific computing* 26, 1 (2004).
M. Meyer, M. Desbrun, P. Schröder, and A. Barr. 2003. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III.* Springer.
D. Mirzaei. 2015. Analysis of moving least squares approximation revisited. *J. Comput. Appl. Math.* 282 (2015), 237–250.
M. Muller. 1956. Some Continuous Monte Carlo Methods for the Dirichlet Problem. *Ann. Math. Statist.* 27, 3 (09 1956), 569–589.
A. Nealen. 2004. An As-Short-as-Possible Intro to Moving Least Squares. (2004). http://www.nealen.com/projects/mls/asapmls.pdf
N. Newton. 1994. Variance reduction for simulated diffusions. *SIAM journal on applied mathematics* 54, 6 (1994), 1780–1805.
B. Øksendal. 2003. Stochastic Differential Equations. In *Stochastic differential equations.* Springer, 65–84.
A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin. 2008. Diffusion curves: a vector representation for smooth-shaded images. In *ACM Trans. Graph.*, Vol. 27. ACM, 92.
A. Pajot, L. Barthe, and M. Paulin. 2011. Sample-Space Bright-Spot Removal Using Density Estimation (regular paper). In *Graphics Interface (GI 2011), St John's, New Found Land (Canada), 25/05/11-27/05/11.* A K Peters, 159–166.
S. Papanicolopulos and A. Zervos. 2013. Polynomial C1 shape functions on the triangle. *Computers & Structures* 118 (2013), 53–58.
E. Pardoux and S. Tang. 1999. Forward-backward stochastic differential equations and quasilinear parabolic PDEs. *Probability Theory and Related Fields* 114, 2 (1999).
S. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, et al. 2010. OptiX: a general purpose ray tracing engine. In *ACM Trans. Graph.*, Vol. 29. ACM, 66.

D. Pavić, M. Campen, and L. Kobbelt. 2010. Hybrid booleans. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 75–87.

J. Peters and U. Reif. 2008. *Subdivision Surfaces*. Springer Berlin Heidelberg, Berlin, Heidelberg, 57–81. https://doi.org/10.1007/978-3-540-76406-9_4

M. Pharr, W. Jakob, and G. Humphreys. 2016. *Physically based rendering: From theory to implementation*. Morgan Kaufmann.

A. Requicha and H. Voelcker. 1977. Constructive solid geometry. (1977).

P. Ribeiro, H. de Campos Velho, and H. Lopes. 2016. Helmholtz-Hodge Decomposition and the Analysis of 2D Vector Field Ensembles. *Comput. Graph.* 55, C (April 2016), 17.

S. Roth. 1982. Ray casting for modeling solids. *Computer graphics and image processing* 18, 2 (1982), 109–144.

F. Rousselle, M. Manzi, and M. Zwicker. 2013. Robust denoising using feature and color information. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 121–130.

B. Schäling. 2014. *The boost C++ libraries*. XML Press.

T. Schneider, Y. Hu, J. Dumas, X. Gao, D. Panozzo, and D. Zorin. 2019. Decoupling Simulation Accuracy from Mesh Quality. *ACM Trans. Graph.* 37, 6 (2019), 280.

S. Sellán, H. Cheng, Y. Ma, M. Dembowski, and A. Jacobson. 2019. Solid Geometry Processing on Deconstructed Domains. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 564–579.

K. Shakenov. 2014. The Solution of the Initial Mixed Boundary Value Problem for Hyperbolic Equations by Monte Carlo and Probability Difference Methods. In *Fourier Analysis*. Springer, 349–355.

E. Shellshear and R. Ytterlid. 2014. Fast Distance Queries for Triangles Lines Points using SSE Instructions. *Journal of Computer Graphics Techniques Vol* 3, 4 (2014).

C. Shen, J. O'Brien, and J. Shewchuk. 2005. Interpolating and Approximating Implicit Surfaces from Polygon Soup. In *ACM Siggraph 2005 Courses*. ACM, 204.

J. Snyder and A. Barr. 1987. Ray Tracing Complex Models Containing Surface Tessellations. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. ACM, New York, NY, USA, 119–128.

O. Stein, E. Grinspun, A. Jacobson, and M. Wardetzky. 2019. A mixed finite element method with piecewise linear elements for the biharmonic equation on surfaces. arXiv:1911.08029 [math.NA]

O. Stein, E. Grinspun, M. Wardetzky, and A. Jacobson. 2018. Natural Boundary Conditions for Smoothing in Geometry Processing. *ACM Trans. Graph.* 37, 2, Article 23 (May 2018), 13 pages. https://doi.org/10.1145/3186564

J. Sun, M. Ovsjanikov, and L. Guibas. 2009. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, Vol. 28. Wiley Online Library, 1383–1392.

T. Sun, P. Thamjaroenporn, and C. Zheng. 2014. Fast multipole representation of diffusion curves and points. *ACM Trans. Graph.* 33, 4 (2014), 53–1.

R. Sutton and A. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT press.

Y. Tong, S. Lombeyda, A. Hirani, and M. Desbrun. 2003. Discrete Multiscale Vector Field Decomposition. *ACM Trans. Graph.* 22, 3 (July 2003), 445–452.

T. Ullrich, V. Settgast, U. Krispel, C. Fünfzig, and D. Fellner. 2007. Distance Calculation between a Point and a Subdivision Surface. In *Vision, Modeling, and Visualization 2007. Proceedings*. Max Planck Institut für Informatik, Saarbrücken, 161–169.

A. Vaxman, M. Campen, O. Diamanti, D. Panozzo, D. Bommes, K. Hildebrandt, and M. Ben-Chen. 2016. Directional Field Synthesis, Design, and Processing. *Computer Graphics Forum* (2016).

E. Veach. 1997. *Robust Monte Carlo Methods for Light Transport Simulation*. Ph.D. Dissertation. Stanford University.

E. Veach and L. Guibas. 1995a. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*. Springer, 145–167.

E. Veach and L. Guibas. 1995b. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM, 419–428.

R. Viertel and B. Osting. 2017. An Approach to Quad Meshing Based on Harmonic Cross-Valued Maps and the Ginzburg-Landau Theory. *SIAM Journal on Scientific Computing* 41 (08 2017).

P. Virtanen, R. Gommers, and Contributors. 2019. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*, Article arXiv:1907.10121 (Jul 2019), arXiv:1907.10121 pages. arXiv:1907.10121

I. Wald, W. Usher, N. Morrical, L. Lediaev, and V. Pascucci. 2019. RTX Beyond Ray Tracing: Exploring the Use of Hardware Ray Tracing Cores for Tet-Mesh Point Location. *Proceedings of High Performance Graphics* (2019).

I. Wald, S. Woop, C. Benthin, G. Johnson, and M. Ernst. 2014. Embree: a kernel framework for efficient CPU ray tracing. *ACM Trans. Graph.* 33, 4 (2014), 143.

H. Wann Jensen. 2001. State of the Art in Monte Carlo Ray Tracing for Realistic Image Synthesis. In *SIGGRAPH Course Notes*.

G. Ward and P. Heckbert. 1992. *Irradiance gradients*. Technical Report. Lawrence Berkeley Lab., CA (United States); Ecole Polytechnique Federale ....

G. Ward, F. Rubinstein, and R. Clear. 1988. A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics* 22, 4 (1988), 85–92.

R. Ytterlid and E. Shellshear. 2015. BVH split strategies for fast distance queries. *Journal of Computer Graphics Techniques (JCGT)* 4 (2015), 1–25.

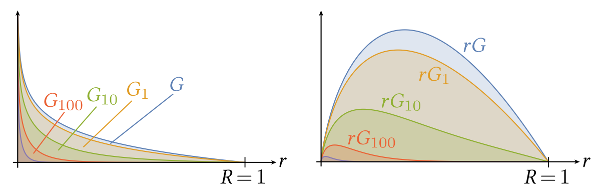


Fig. 28. To importance sample Poisson and screened Poisson equations we need to sample from distributions that depend only on the radius r (here shown for 2D, and for several values of the screening parameter c). Although the Green's functions G have singularities at $r = 0$ *(left)*, the associated radial distributions rG are nonsingular due to a change in measure *(right)*.

Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, H. Shum, and H. Shum. 2004. Mesh Editing with Poisson-based Gradient Field Manipulation. In *ACM Trans. Graph.*, Vol. 23. ACM, 644–651.

R. Zhao, M. Desbrun, G. Wei, and Y. Tong. 2019. 3D Hodge Decompositions of Edge- and Face-based Vector Fields. *ACM Trans. Graph.* 38, 5 (2019).

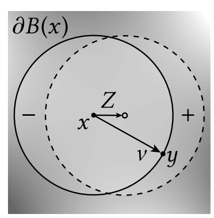
Q. Zhou, E. Grinspun, D. Zorin, and A. Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Trans. Graph.* 35, 4 (2016), 1–15.

Q. Zhou and A. Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).

Q. Zhu, L. Hernquist, and Y. Li. 2015. Numerical convergence in smoothed particle hydrodynamics. *The Astrophysical Journal* 800, 1 (2015), 6.

M. Zwicker, W. Jarosz, J. Lehtinen, B. Moon, R. Ramamoorthi, F. Rousselle, P. Sen, C. Soler, and S. Yoon. 2015. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports)* 34, 2 (May 2015), 667–681.

A INTEGRAL FORMULAS FOR DERIVATIVES



A.0.1 Gradient of Boundary Term. The mean value property from Eqn. 4 can be restated by saying that the value of u at any point x is equal to its average value over the interior of any ball $B(x) \subset \Omega$, *i.e.*, $u(x) = \frac{1}{|B(x)|} \int_{B(x)} u(y)\, dy$. If we move the ball in some direction Z, the change in u is therefore given by

$$\langle \nabla u(x), Z \rangle = \frac{1}{|B(x)|} \int_{\partial B(x)} \langle Z, \nu(y) \rangle u(y)\, dy,$$

where ν is the outward unit normal of $\partial B(x)$. Intuitively: if we shift the ball slightly, we pick up a contribution in the "front" and lose a contribution in the "back," both proportional to the magnitude of u. Since this relationship holds for all Z, and since $\nu = (y-x)/|y-x| = (y-x)/R$, we obtain the expression from Eqn. 12.

This expression can also be derived using the framework of *Malliavin calculus* [Bell 2012], which instead approaches the derivation from the viewpoint of deviations of stochastic processes (in particular, Wiener processes). We used this machinery to derive the boundary Hessian term appearing in Sec. 3.3.

B SAMPLING DISTRIBUTIONS

We here give the Green's functions needed to estimate solution values and derivatives for the PDEs in Sec. 2. Since these PDEs are isotropic, their Green's functions can be expressed purely in terms of the distance $r := |y - x|$ from the center of the ball $B(x)$ (see Fig. 28). Derivations can be found in Koshlyakov et al. [1964].

To sample from the probability distribution $p_G := G/\int_B G$ associated with a Green's function G, we first pick a unit direction $\hat{y}$ uniformly on the unit sphere [Arvo 2001], then sample the radius from a distribution proportional to rG in 2D, or $r^2 \sin\theta$ in 3D, where θ is the polar angle of $\hat{y}$ in spherical coordinates. The extra factor in front of G accounts for the change of measure between polar and Cartesian coordinates, and eliminates the singularity at $r = 0$ (see Fig. 28). The final sample point is then $y = r\hat{y}$. When uniformly sampling the source contribution (as in Sec. 2.3), or when importance sampling source terms (Sec. 4.2.2), samples of G near $r = 0$ could perhaps cause "bright spots" [Pajot et al. 2011], though in practice we do not observe such behavior.

B.1 Harmonic Green's Function

For a ball B of radius R, we use G to denote the *harmonic Green's function* of the Laplace operator on B, with Dirichlet boundary conditions. In two and three dimensions, we have

$$G^{2D}(x,y) = \tfrac{1}{2\pi}\log(R/r), \qquad G^{3D}(x,y) = \tfrac{1}{4\pi}\tfrac{(R-r)}{rR},$$

resp., where $r = |y - x|$. The integrals of these functions over the ball (needed to obtain the associated probability distributions p_G) are given by

$$\int_{B(x)} G^{2D}(x,y)\,dy = \tfrac{R^2}{4}, \qquad \int_{B(x)} G^{3D}(x,y)\,dy = \tfrac{R^2}{6},$$

and their gradients with respect to x are given by

$$\nabla G^{2D}(x,y) = \tfrac{y-x}{2\pi}\left(\tfrac{1}{r^2} - \tfrac{1}{R^2}\right), \qquad \nabla G^{3D}(x,y) = \tfrac{y-x}{4\pi}\left(\tfrac{1}{r^3} - \tfrac{1}{R^3}\right).$$

B.2 Yukawa Potential

The Green's function G_c for a screened Poisson equation depends on the parameter c appearing in Eqn. 9. It is given by the so-called *Yukawa potentials* [Elepov and Mikhailov 1969]

$$G_c^{2D}(x,y) = \tfrac{1}{2\pi}\left(K_0(r\sqrt{c}) - I_0(r\sqrt{c})\tfrac{K_0(R\sqrt{c})}{I_0(R\sqrt{c})}\right),$$

$$G_c^{3D}(x,y) = \tfrac{1}{4\pi}\left(\tfrac{\sinh((R-r)\sqrt{c})}{r\sinh(R\sqrt{c})}\right),$$

which have integrals

$$\int_{B(x)} G_c^{2D}(x,y)\,dy = \tfrac{1}{c}\left(1 - \tfrac{1}{I_0(R\sqrt{c})}\right),$$

$$\int_{B(x)} G_c^{3D}(x,y)\,dy = \tfrac{1}{c}\left(1 - \tfrac{R\sqrt{c}}{\sinh(R\sqrt{c})}\right),$$

and gradients

$$\nabla G_c^{2D}(x,y) = \tfrac{(y-x)\sqrt{c}}{2\pi}\Bigg(\tfrac{K_1(r\sqrt{c})}{r} - \tfrac{K_1(R\sqrt{c})}{R}\tfrac{I_0(r\sqrt{c})}{I_0(R\sqrt{c})} + \tfrac{K_0(R\sqrt{c})}{I_0(R\sqrt{c})}\left(\tfrac{I_1(r\sqrt{c})}{r} - \tfrac{I_1(R\sqrt{c})}{R}\tfrac{I_0(r\sqrt{c})}{I_0(R\sqrt{c})}\right)\Bigg),$$

$$\nabla G_c^{3D}(x,y) = \tfrac{y-x}{4\pi}\Bigg(\tfrac{\sqrt{c}\cosh((R-r)\sqrt{c})}{r\sinh(R\sqrt{c})}\left(\tfrac{1}{r} - \tfrac{1}{R}\right) + \tfrac{\sinh((R-r)\sqrt{c})}{r\sinh(R\sqrt{c})}\left(\tfrac{1}{r^2} + \tfrac{\sqrt{c}\cosh(R\sqrt{c})}{R\ \sinh(R\sqrt{c})}\right)\Bigg).$$

The functions I_0, I_1 and K_0, K_1 are modified Bessel functions of the first and second kind respectively. Routines to efficiently evaluate these functions are commonly available in numerical libraries such as *Boost* [Schäling 2014] and *SciPy* [Virtanen et al. 2019].

B.2.1 Normalization Constants. The estimator for the screened Poisson equation is the same as for the ordinary Poisson equation, except that the term $\hat{u}_f(x_{k_1})$ in Eqn. 8 is multiplied by the factors

$$C^{2D} := \frac{1}{I_0(R\sqrt{c})} \qquad \text{and} \qquad C^{3D} := \frac{R\sqrt{c}}{\sinh(R\sqrt{c})},$$

in 2D and 3D (*resp.*), where c is the constant from Eqn. 9. See for instance Booth [1981, Equation 13].

B.2.2 Terms for Hessian Estimator. In two and three dimensions, the functions ψ and ϕ associated with the harmonic Green's functions for a ball of radius R are given by

$$\psi^{2D}(x,y) = \tfrac{1}{2\pi}\tfrac{2}{R^4}, \qquad \phi^{2D}(x,y) = \tfrac{1}{2\pi}\tfrac{r^2-R^2}{R^4},$$

$$\psi^{3D}(x,y) = \tfrac{1}{4\pi}\tfrac{3}{R^5}, \qquad \phi^{3D}(x,y) = \tfrac{1}{4\pi}\tfrac{r^2-R^2}{R^5},$$

resp., where $r = |y - x|$.

Received January 2020